INP ENSEEIHT

N° d'ordre : 2300

Institut National
Polytechnique de Toulouse

École Doctorale
Informatique et Télécomminucation

Agents Mobiles Coopérants pour les Environnements Dynamiques

THÈSE

présentée et soutenue publiquement le 2 décembre 2005

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Toulouse
(ENSEEIHT)

par

Christophe CUBAT DIT CROS

Composition du jury

Président	Daniel	HAGIMONT	INP Toulouse (N7)
Rapporteurs	Nicole	LEVY	Université de Versaille
	Frédéric	GUINAND	Université du Havre
Examinateur	Franck	BARBIER	UPPA (Pau)
Co-encadrant	Philippe	QUEINNEC	INP Toulouse (N7)
Directeur de thèse	Gérard	PADIOU	INP Toulouse (N7)

Institut de Recherche en Informatique de Toulouse — UMR 5505

IRIT Site de l'ENSEEIHT

Mise en page à partir de la classe **thloria**
et à l'aide du livre « LaTeX par la pratique » [Rol99]

« L'impossible appartient à ceux qui ne croient pas en eux »

À vous, Nélia et Julien,
mes tous jeunes nièce et filleul,
puissions-nous vous guider à dépasser
ce qui nous a été transmis.

Remerciements

La réalisation d'un doctorat est une tranche de vie à part entière qui laisse paraître un profond égoïsme en fin de parcours mais qui s'est nourrie, tout du long, de nombreuses et diverses influences. Beaucoup de personnes participent directement ou indirectement à sa concrétisation et méritent d'être remerciés quel qu'en soit leur degré d'investissement. Malheureusement, ici tout le monde ne peut apparaître nommément et je me dois de faire une sélection subjective.

D'un point de vue professionnel, je commencerai par remercier mon directeur de thèse *Gérard PADIOU* et mon co-encadrant *Philippe QUEINNEC* pour le travail qu'ils ont effectué à mes côtés et qui était emprunt tout autant de conseils pour orienter mes recherches que de critiques nécessaires à la pertinence de mon étude.

Ensuite, je souhaite remercier mes rapporteurs *Nicole LEVY* et *Frédéric GUINAND* pour avoir accepté d'évaluer mon travail et pour les commentaires qu'ils ont formulés afin de le perfectionner. Je tiens aussi à exprimer ma gratitude envers *Franck BARBIER* qui a eu la gentillesse de lire l'intégralité de mon manuscrit et qui a usé de son expertise pour déceler certains aspects qui m'avaient échappés.

Je tiens aussi à remercier les nombreux collègues qui ont partagé mon quotidien. En particulier, *Xavier CRÉGUT* pour avoir été mon premier partenaire de bureau et pour avoir partagé sa science de l'enseignement. *Marc PANTEL* pour s'être intéressé à mon travail avec un point de vue différent. Et aussi, *Daniel HAGIMONT*, le dernier arrivant, pour s'être très largement penché sur la préparation de ma soutenance et pour avoir accepté d'être président de mon jury.

Je n'oublierai pas *Annie ADELENTADO, Viollette ANTON, Anne-Marie ZERR* et *Stéphanie CHAP*, les responsables des différents secrétariats, pour leur bonne humeur et la facilité avec laquelle elles ont réglé mes problèmes administratifs. J'ajouterai sans aucune retenue *Bernie SANTONI* et *Agnès CARADONNA*, les responsables des formations du CIES, avec qui j'ai partagé de bien agréables colloques.

Je n'oublierai pas non plus les différents doctorants de l'INPT qui m'ont permis de faire régulièrement une « sINPThèse » de mon sujet. Je citerais en particulier ceux qui sont devenus de véritables amis : *Patoche* et *Polo* qui m'ont montré la voie permettant de conclure ma vie étudiante, ainsi qu'*Ariel* et *Clovis*, qui m'ont fait passer d'excellents instants aux moments où la rédaction se faisait des plus pesante.

D'un côté plus personnel, je ne saurais montrer trop de reconnaissance aux membres de ma famille. A mes parents dans un premier lieu, *Anne* et *Jean* , qui ont toujours su m'accorder leur confiance et leur soutien dans les choix parfois difficiles que j'ai pu faire durant mes études. A mes deux frères ensuite, *Sébastien* et *Frédéric*, qui ont été capables de me montrer le pragmatisme qui parfois manque dans les domaines scientifiques. Enfin, à ma belle sœur et à ma nièce, *Karine* et *Nélia*, qui au travers de leur fraîche arrivée savent apporter une touche féminine que j'apprécie tant. Famille, voyez dans cet ouvrage le témoignage des profonds sentiments que je vous porte.

Ensuite j'aimerais mettre en avant tous mes amis, comparses de soirée et ceux avec qui je passe d'agréables moments de détente. *Jean-Michel*, frère de coeur, et sa (ma) famille (de coeur) qui me suivent depuis tant d'années et qui sait toujours trouver les mots qui me touchent au plus profond. *Peyo* qui me témoigne sans détour son amitié et m'apporte continuellement son soutien. *Audrey, Céline, Jérome, Cédric* et *Miguel*, les membres de la bande paloise qui m'ont toujours fait passer des mois de juin inoubliables. *Patrick* un récent heureux papa qui s'intéresse

sans cesse à mes travaux sans jamais oublié la convivialité. *Isaac, Fabienne, Franck, Flore, Doumé* et autres qui partagent ma passion de la danse au fil de soirées sans fin. A vous tous merci, sans ces moments de connivence et votre confiance en moi, je n'aurais sans doute jamais mené à bien cet ouvrage.

Je tiens à terminer ces remerciements en adressant de très fortes pensées à ma cousine *Maud LESCOFFIT* qui a su mettre à profit son expérience d'écrivain pour trouver les mots me permettant de lancer la phase rédactionnelle sur de bons rails. Cousine, tu sais d'où je viens et c'est en grande partie grâce à toi que tu me verras atteindre les objectifs que je me suis fixés. A jamais merci pour me pousser à toujours dépasser mes limites.

Table des matières

Introduction

Contexte de l'étude

Lorsque l'on utilise le terme « *informatique* », on peut entendre plusieurs sens allant du logiciel de bureautique (traitement de texte, tableur ...) jusqu'au composant d'un ordinateur (processeur, mémoire etc..). Dans notre cas, nous nous intéressons à l'informatique répartie qui se définit par la prise en compte de la mise en relation des ordinateurs grâce à un réseau de communication type Internet. Dans ce domaine, nous pouvons classifier deux champs d'investigation : le premier est le niveau matériel qui cherche à trouver les techniques permettant de mettre en relation les machines, et le second est le niveau logiciel qui cherche à concevoir des programmes capables d'exploiter la répartition des ordinateurs.

D'un point de vue matériel, les technologies des réseaux de communication sont construites, jusqu'à présent, sur des architectures de communication quasiment fixes, ayant éventuellement une topologie particulière (anneau, bus, arbre par exemple) et dans lesquelles l'évolution des connexions (apparition/rupture) était un phénomène rare. Cependant, avec l'utilisation de plus en plus courante des technologies sans fil, les réseaux évoluent de plus en plus vers de nouvelles architectures dynamiques à large échelle. Les sites deviennent mobiles, apparaissent et disparaissent fréquemment, les connexions évoluent de façon continue. Le domaine des réseaux *ad hoc*, qui permettent la mise en relation de machines à la volée, a conduit à de nombreuses études sur la gestion du dynamisme : à titre d'exemple, la notion de groupe de processus bien connue de l'algorithmique répartie, a dû être reconsidérée pour tenir compte de l'évolution du voisinage de communication des sites hôtes.

D'un point de vue logiciel, ces nouvelles architectures ont amené et amènent aujourd'hui encore à reconsidérer les systèmes et applications réparties qui étaient conçus selon les anciennes structures fixes. La conception des systèmes et applications répartis repose sur un ensemble de concepts et d'abstractions tels que le client-serveur, le pair à pair ou le schéma de publication/abonnement. Parmi ces propositions, la notion d'agent mobile constitue une abstraction déjà ancienne mais présentée parfois comme une approche pouvant apporter des performances meilleures par rapport au schéma de base client/serveur : la mobilité de l'agent lui permet de déporter par exemple les traitements au plus près des données et éviter ainsi de coûteux transferts de données. Cependant, aucune application majeure n'est venue confirmer la supériorité des agents par rapport aux autres approches, les comparaisons ayant été menées dans des architectures de réseau fixes.

De nombreux environnements d'agents mobiles ont été proposés et leur diversité a sans doute nui à leur utilisation dans le milieu industriel. Dans la plupart des cas, les applications

ont fait appel à des agents de granularité élevée (agents exécutant des traitements relativement complexes et des codes importants), de faible mobilité (peu de migrations pour accomplir un traitement global) et en nombre relativement restreint (un agent seul exécute une requête complète en coopérant éventuellement avec quelques agents). Les réseaux dynamiques, par exemple les réseaux de capteurs, ont donné lieu à quelques études utilisant des agents de granularité beaucoup plus faible (peu de code, traitements simples), de mobilité forte (migrations rapides et nombreuses) et d'un nombre élevé d'agents. Il semble donc intéressant de reconsidérer l'apport des agents mobiles dans toute leur acceptation (quel que soit leur granularité, mobilité et échelle) dans le contexte des réseaux dynamiques. Bien qu'un tel choix implique la présence et la maîtrise d'une double mobilité, celle, physique, des sites et celle, logique, des agents, les propriétés intrinsèques des agents méritent d'envisager leur utilisation dans un tel contexte.

Thèse soutenue

Nous pensons que le dynamisme physique apporté par le déplacement des unités mobiles et le souhait des utilisateurs nomades d'accéder à Internet n'importe où et n'importe quand, vont imposer un fort dynamisme matériel. Dans le cadre classique des structures fixes, le système prend en charge la gestion de la répartition afin de la masquer aux concepteurs des applications distribuées. Dans notre cas, avec les architectures dynamiques, le système aura de grandes difficultés à masquer la répartition qui devra être prise en compte par le concepteur. Dans de tels contextes, nous pensons que les propriétés des agents mobiles sont les mieux adaptées aux changements fréquents de ces architectures dynamiques à grande échelle.

Cependant, la double mobilité, logicielle et matérielle, ne va pas être facile à gérer pour un concepteur qui souhaite utiliser un service réparti. En effet, il devra être capable de déterminer sa localisation et le moyen de l'utiliser dans un environnement en constante évolution. Une communication distante paraissant difficile à réaliser, nous pensons qu'il faut privilégier les interactions locales. Nous proposons donc d'offrir au concepteur un service système qui prenne en charge une partie de la répartition afin de l'aider lors de la détermination de la localisation d'un service et du chemin permettant de mettre en place l'interaction locale.

Pour réaliser ce service, nous proposons de construire une couche dite *d'ambiance* à laquelle s'adressent les agents cherchant un service précis. Elle sera composée d'un nombre élevé d'agents ayant une granularité faible, de fréquents et rapides déplacements, s'ignorant et ne commiquant pas directement entre eux. Ces caractéristiques doivent leur permettre de trouver toutes les ressources présentes dans l'environnement et d'en constater les changements, les évolutions. Le concepteur se sert alors de la couche d'ambiance pour obtenir des informations sur une ressource recherchée.

Plan du mémoire

La première partie de cette thèse s'intéresse à l'état de l'art du contexte des agents mobiles et s'articule en deux chapitres. Le premier étudie les différentes méthodes existantes pour construire et utiliser les réseaux de communication. En commençant par le point de vue matériel, nous présentons les différentes méthodes pour la construction des réseaux d'ordinateurs en insistant particulièrement sur les technologies sans fil. Puis, nous regardons le côté logiciel afin d'étudier les différentes approches de conception existant pour construire les applications

réparties et particulièrement celles utilisant la mobilité. Ensuite nous étudions les avantages et inconvénients des agents mobiles et, enfin, nous en précisons l'usage.

Dans le second chapitre, nous étudions plus précisément le domaine de recherche des agents mobiles. Pour cela, nous commençons par une étude comparative des différentes plates-formes existantes. Ensuite, nous détaillons les modes d'interaction existant pour les agents mobiles. Nous terminons par les différents domaines d'application envisageables permettant d'utiliser au mieux les caractéristiques des agents mobiles.

La seconde partie du mémoire porte sur notre modèle d'agents mobiles coopérants. Nous consacrons le troisième chapitre à la description du problème de la double mobilité à large échelle et à la caractérisation des types d'agents présents dans un environnement dynamique. Ceux-ci nous amènent à proposer une solution pour la gestion de la double mobilité grâce à la mise en place d'une couche dite d'ambiance. Le quatrième chapitre est consacré à l'étude plus précise de la prise en charge de la gestion de la répartition dans des environnements dynamiques grâce à l'utilisation des agents mobiles. Le cinquième chapitre constitue la description du modèle d'agent mobile coopérant comprenant une description précise d'un agent coopérant, des services de base du système et du mode de visite d'un site.

La dernière partie de ce document est consacrée à l'expérimentation de notre modèle d'agent mobile coopérant. À partir du simulateur que nous avons réalisé et exposé dans l'annexe A, nous dédions le sixième chapitre à l'exposition des résultats obtenus sur les différentes expérimentations que nous avons menées. Nous présentons, dans un premier temps, les études menées sur la couche d'ambiance afin de déterminer son adéquation avec la double mobilité. Ensuite, nous présentons plusieurs applications construites sur notre modèle afin de permettre leur adaptation dans les environnements dynamiques.

Nous concluons par un bilan sur la double mobilité, plus précisément sur les caractéristiques apportées par les agents mobiles dans les environnements dynamiques, suivi des perspectives de recherche pouvant faire suite à ce travail.

Première partie

État de l'art

Du statique à la mobilité autonome

Réseau dynamique et mobilité logicielle

1

Dans ce premier chapitre, nous décrivons le contexte général de notre étude portant sur les agents mobiles coopérants. De manière classique, l'exécution d'un logiciel s'effectue de façon isolée, *i.e* sans avoir besoin de communiquer avec d'autres logiciels, et sur une unique machine, *i.e* le logiciel dispose localement de tous les éléments nécessaires. Un traitement de texte est un bon exemple de ce type de logiciel. Dans le contexte où nous nous situons ici, les applications sont dites « réparties » car elles vont prendre en compte les relations existantes entre les machines appartenant à des *réseaux d'ordinateurs*. Ces applications réparties peuvent, alors, s'exécuter sur plusieurs machines ou peuvent récupérer les éléments distribués dont elles ont besoin. Un navigateur Web répond à ce type d'applications.

Les interconnexions de machines, appelées *réseaux d'ordinateurs*, ont été introduites au départ pour faciliter les échanges de données entre les ordinateurs sans utiliser de fastidieux supports de stockage (bandes, disquettes, cartes perforées etc.) et pour permettre la mise en relation des usagers séparés par de longues distances [Tan90]. À l'origine, ces réseaux formaient des structures de communication indépendantes, mais pour permettre aux utilisateurs d'avoir un environnement homogène, les techniques d'*interconnexion de réseaux* se sont largement répandues afin d'offrir un système affranchi des problèmes d'hétérogénéité matérielle [Com96]. Ces systèmes, généralement appelés un *internet* se sont largement développés jusqu'à former le fameux *Internet* constituant le *World Wide Web*. La première partie de ce chapitre porte sur l'étude des différentes architectures existantes dans ces interconnexions et plus particulièrement sur les technologies sans fil.

Pour tirer partie au mieux de ces environnements connectés, le développement d'applications pouvant prendre en compte un ensemble de machines a été rapidement étudié. C'est ce type d'applications que nous désignons par *réparties* ou *distribuées*. Celles-ci peuvent s'exécuter sur plusieurs machines différentes, utiliser des ressources distribuées, mettre en place des mécanismes de tolérance aux fautes, etc.. Ce qui permet d'aller bien au-delà du simple échange de données. L'étude des différentes méthodes existantes, pour la conception des applications distribuées, est l'objet de la seconde partie du chapitre.

Les deux dernières parties de ce chapitre portent plus spécifiquement sur l'approche qui nous intéresse dans ce document, à savoir les agents mobiles. Nous exposerons principalement leurs avantages et inconvénients par rapport aux mécanismes de conception plus classiques et enfin nous apporterons quelques précisions sur la notion d'agent qui est aussi employée dans d'autres domaines de recherche.

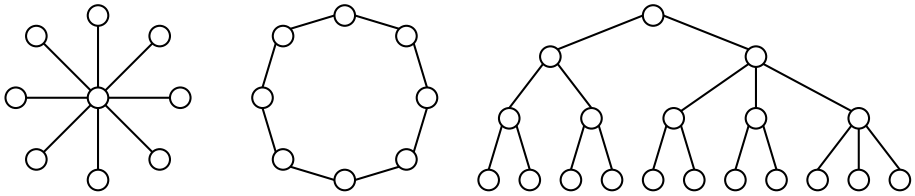


FIG. 1.1 – Exemple de topologie réseau filaire

1.1 Le contexte physique : l'architecture réseaux

Dans cette partie nous présentons le contexte physique servant de base à l'exécution des applications réparties. Les interconnexions d'ordinateurs sont construites selon certains critères de performance qui vont influencer la conception et le déroulement des applications. Cette influence va se révéler très importante dans les environnements sans fil où les liens physiques sont très versatiles. Nous allons donc commencer par présenter les schémas d'organisations existants pour les réseaux filaires qui vont nous servir de référence, par la suite, pour la description des architectures sans fil.

1.1.1 Les réseaux filaires

À l'heure actuelle, les interconnexions d'ordinateurs les plus répandues sont toujours construites sur des mises en relation de machines grâce à l'utilisation de câbles classiques. Les réseaux filaires, ainsi formés, peuvent être construits selon plusieurs architectures en fonction de besoins bien spécifiques. Nous pouvons citer, entre autre, les topologies étoile, anneau ou encore arbre (cf fig 1.1). Cependant cette multitude de possibilités de construction ne facilite pas la tâche aux concepteurs d'applications. C'est pour cette raison qu'un homogénéisation a été introduite grâce à la mise en place de la norme TCP/IP (*Transmission Control Protocol et Internet Protocol*)

Cette norme permet d'offrir au concepteur un environnement homogène, d'un point de vue matériel, construit sur un ensemble de services auxquels s'adressent les applications distribuées. Cette technique propose de construire un internet sur l'utilisation d'appareils permettant de mettre en relation deux ou plusieurs sous-réseaux. La fonction de cet appareil, appelé routeur, étant de faire transiter les communications entre les différents sous réseaux. Ceux-ci restent indépendants et n'ont pas besoin de connaître les autres architectures. En répétant ces interconnexions, on obtient alors des environnements interconnectés comme le montre la figure 1.2

Le principal intérêt de cette technique d'un point de vue des ordinateurs hôtes, *i.e* ne participant pas à la fonction de routage, c'est qu'elle propose une abstraction de l'environnement physique pour la gestion de la communication. En d'autres termes, lorsqu'un ordinateur hôte souhaite communiquer avec un autre ordinateur hôte, il perçoit l'internet comme une espèce de nébuleuse qui va prendre en charge la mise en relation et la gestion des communications avec le partenaire cible. Pour mettre en place cette gestion, l'environnement dispose d'un ensemble de services systèmes permettant de connaître, de référencer et de communiquer sans ambiguïté avec tout élément présent dans l'environnement. . Les concepteurs d'applications distribuées disposent alors d'un environnement homogène permettant d'établir facilement des

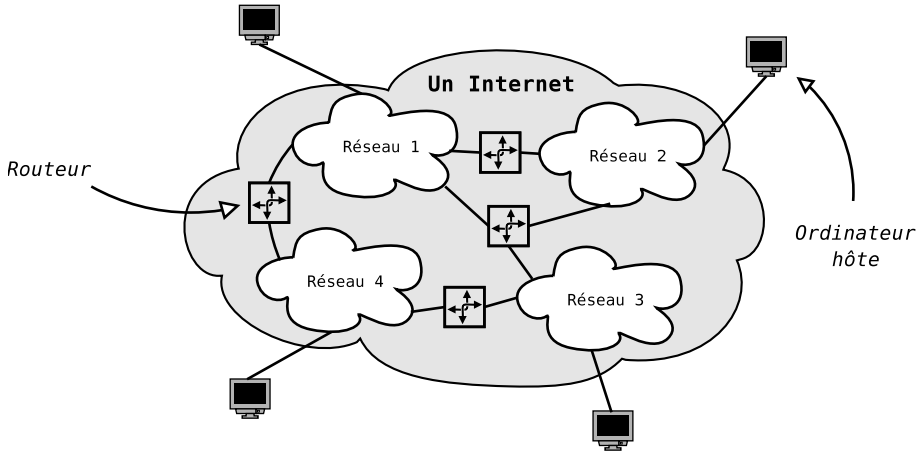


FIG. 1.2 – Architecture d'un Internet

communications distantes entre les différents ordinateurs hôtes. C'est ce qu'on appelle un *réseau*.

Dans la plupart des cas, les services composants un internet sont gérés de manière centralisée et hiérarchique. À chaque sous-réseau correspond un gestionnaire capable de gérer les communications entre les éléments dont il a la charge. Si la cible d'une communication n'appartient pas à son sous-réseau, il s'adresse alors à un élément d'un « *niveau supérieur* » qui prendra le relais pour mettre en place et gérer la communication avec la cible. Cette méthode centralisée convient parfaitement aux environnements filaires peu enclins au dynamisme physique. En effet, de par la nature de la construction des réseaux filaires, le dynamisme de ce type d'environnement se cantonne aux pannes systèmes, à de fausses manipulations humaines (débranchement d'un câble) ou à l'intégration d'unités mobiles dans un cadre très strict et limité. Ces événements étant assez rares et bien définis, la gestion centralisée et hiérarchique reste parfaitement adaptée.

Avec ces différentes caractéristiques d'homogénéité et de stabilité physique, nous disposons, d'un point de vue conception, d'un système qui permet d'exécuter des applications utilisant des communications distantes qui pourront se dérouler sans interruption. Nous verrons que cette caractéristique est essentielle pour certains schémas d'organisation d'applications réparties.

Cependant l'apparition des unités mobiles pouvant se déplacer de sous-réseau en sous-réseau au sein d'un même internet va remettre en cause la vision statique de l'environnement physique. L'utilisation de plus en plus massive des ordinateurs mobiles est prise en compte dans la prochaine version de norme de la technologie TCP/IP, la version 6 (IPv6) [Ciz98], grâce à l'introduction de MOBILE-IP qui permet de rendre transparents, aux éventuels partenaires, les déplacements d'un ordinateur hôte. Cette solution a surtout été pensée pour les cas où les déplacements dans un autre sous-réseau, que celui d'origine, restent rares et d'une période assez longue. On peut prendre l'exemple d'un employé d'une entreprise en mission de longue durée chez un de ses clients. Nous verrons que cette technique s'accorde mal avec un dynamisme bien plus important (cf chap. 4).

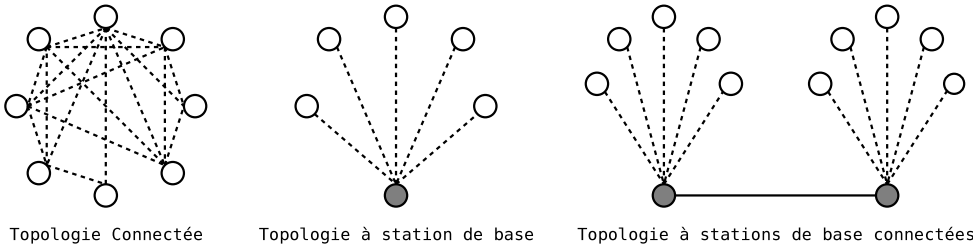


FIG. 1.3 – Topologies réseaux sans fil

1.1.2 Les réseaux sans fil

Les unités mobiles, tel les ordinateurs portables ou les téléphones, vont essentiellement utiliser des technologies sans fil pour se connecter aux réseaux et vont apparaître comme n'importe quel autre ordinateur hôte. En effet, avec la chute des prix du matériel de transmission sans fil, avec l'augmentation de l'autonomie des batteries et enfin avec l'augmentation des performances de la miniaturisation des processeurs de forte puissance, nous avons aujourd'hui des unités mobiles de petite taille, tel les assistants numériques (PDA) ou encore les smartphones, capables de se déplacer facilement avec leur propriétaire tout en exécutant des applications complexes et en ayant la capacité de rester connectées à un réseau sans fil.

Cependant l'utilisation de ce type de matériel va imposer la mise en place de réseaux sans fil avec de nouvelles topologies qui devront être prises en compte lors de leur intégration dans un internet. Cette prise en compte se fera plus précisément au niveau des services systèmes. La capacité propre des unités mobiles à se déplacer tout en voulant conserver une connexion à un réseau de communication va demander de reconsidérer la manière de construire les internets. En nous basant sur la thèse [Lao02], nous proposons les différentes topologies que nous pouvons trouver dans les réseaux sans fil, puis nous regardons la méthode dite *ad hoc* pour la constitution d'un réseau et nous terminons sur notre vision de l'utilisation à venir des technologies sans fil.

Topologies existantes

Les différentes topologies existantes que nous pouvons trouver dans un environnement sans fil sont résumées sur la figure 1.3 et sont détaillées comme suit :

Topologie connectée Ce type de topologie est la plus simple à mettre en place car les éléments d'un réseau vont communiquer directement deux à deux. Cependant, il s'agit aussi de la plus fragile car si un lien venait à disparaître entre deux éléments, il n'y aurait aucun moyen de poursuivre leur communication.

Topologie à station de base Dans cette technique, une station de base va concentrer toutes les communications du réseau. Ainsi, l'unique moyen de communiquer entre deux éléments est de passer par la station de base. Cette centralisation fragilise fortement le réseau qui ne pourra plus fonctionner si la station de base vient à disparaître, suite à une panne par exemple.

FIG. 1.4 – Réseau Ad Hoc

Topologie combinée Dans cette topologie, on utilise une relation filaire entre deux stations de base. Si les deux premières techniques servent principalement à construire des réseaux autonomes, celle-ci permet de mettre en place un internet en offrant la possibilité d'établir des communications entre les unités sans fil de deux sous-réseaux différents. Les stations de base peuvent aussi assurer la fonction de routeur. C'est le principe utilisé par les réseaux téléphoniques GSM.

À l'heure actuelle, la topologie combinée est la plus largement répandue surtout grâce à l'utilisation des bornes d'accès WiFi (norme IEEE 802.11 [CWKS97]). En effet, de plus en plus d'*hotspot* apparaissent dans les lieux publiques, les entreprises, les campus ou encore à domicile. Ce succès est principalement dû au faible coût nécessaire pour étendre un réseau prenant en compte les unités mobiles et proposant ainsi un accès sans fil à Internet pour un ensemble de clients.

Cependant, les topologies combinées restent basées sur l'utilisation d'une station de base qui dispose d'un nombre limité de connexions et d'une bande passante à partager entre tous les éléments en cours de connexion. Ces deux limites associées à la fragilité de la centralisation, due à l'utilisation d'une borne d'accès, ne permettent pas de croire que la topologie combinée puissent suffire, à terme, pour gérer les lieux de fortes fréquentations (gare, aéroport, etc.) ou pour satisfaire les besoins de forte mobilité des utilisateurs qui demanderaient de trop fréquentes reconfigurations.

Les réseaux *ad hoc*

Pour permettre de complètement décentraliser la construction d'un réseau de machine sans fil, on utilise les topologies dites *ad hoc*. On qualifie un réseau par le terme « ad hoc » lorsqu'il est constitué d'un ensemble d'éléments autonomes et qu'il est construit sans aucune infrastructure existante. En d'autres termes, lorsque deux ou plusieurs machines sans fil se situent dans un même lieu, elles peuvent construire un réseau directement entre elles sans avoir recours à un cadre déjà établi, comme avec l'utilisation d'une borne d'accès. Une exemple d'un tel réseau est illustré par la figure 1.4.

La particularité de ce genre de topologie, par rapport aux topologies connectées, c'est que chaque élément participe à l'architecture générale en relayant le trafic vers les différents éléments directement joignables, *i.e* appartenant à son rayon d'émission. Ces éléments constituent le *voisinage* d'un noeud. Ce voisinage n'est pas constant dans le temps, bien au contraire, il va

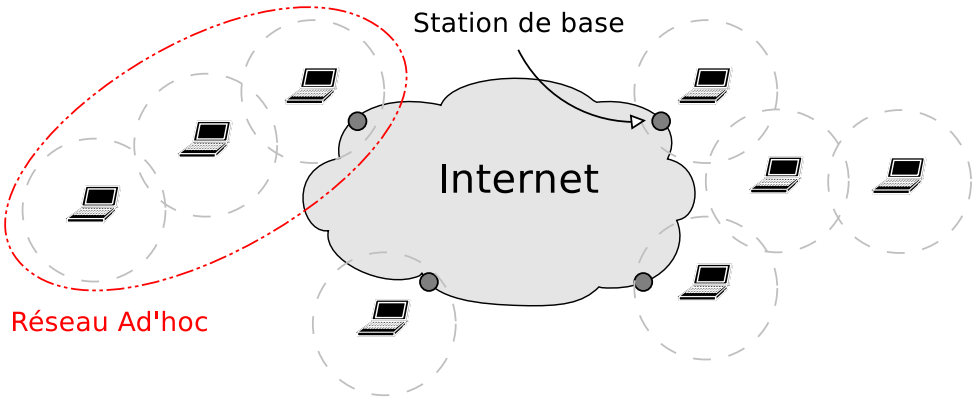


FIG. 1.5 – Réseau Hybride

continuellement évoluer en fonction des déplacements des unités mobiles, ce qui va induire une topologie dynamique.

Une propriété intéressante des topologies *ad hoc* est la totale décentralisation de la gestion du système due aux éléments autonomes. Ainsi les déplacements des unités mobiles sont gérés de façon standard en mettant à jour le voisinage des noeuds et en relayant le trafic entre tous les membres d'un même voisinage. Cette faculté apporte un grande résistance aux fautes pouvant facilement apparaître dans les environnements sans fil, comme la disparition d'un noeud.

Plusieurs études sont menées sur les réseaux *ad hoc* [KTI99, VB00] afin de faciliter leurs développements. Celle qui nous intéresse particulièrement est menée par le groupe de travail MANET de l'IETF [CM99]. Celui-ci cherche à intégrer la technologie TCP/IP [LA03] directement aux réseaux *ad hoc* afin de permettre une extension d'un internet classique et d'offrir un environnement homogène pour la conception d'applications distribuées.

1.1.3 Conclusion : vers une architecture hybride

Les différentes techniques, que nous venons de présenter pour créer des architectures sans fil, vont pouvoir s'appliquer dans un certain nombre de domaines où il sera plus coûteux ou plus difficile de déployer une architecture filaire classique. Par exemple dans des milieux hostiles (champ de bataille, cratère de volcan ...) [HMW98], lors de l'intervention de secours suite à une destruction de l'architecture filaire (catastrophe naturelle, attentat ...), ou bien plus simplement le long des autoroutes, sur un campus, ou bien encore pour des utilisations temporaires comme des congrès, des manifestations sportives. Tous ces domaines d'applications peuvent, en fonction des besoins, se servir des différentes topologies, *i.e* connectées, station de base ou combinées.

Cependant, les défauts inhérents aux technologies sans fil, basées principalement sur les transmissions radio, ne vont pas s'accorder avec une gestion centralisée. Parmi ces défauts, nous pouvons énoncer entre autres : les interférences, les débits faibles, les liens versatiles, la portée radio limitée et surtout le déplacement des unités mobiles induisant une topologie changeante. Ainsi, la mise en place de topologies totalement centralisées par l'utilisation de station de base semble peu adaptée. Pour permettre de pallier à ces problèmes, il serait plus intéressant de recourir à un mode de connexion *ad hoc*.

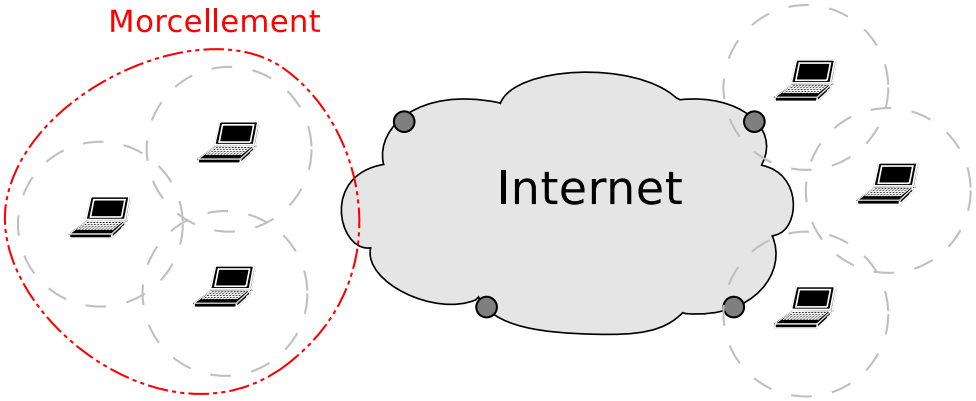


FIG. 1.6 – Morcellement de réseau

Les topologies *ad hoc* permettent une totale décentralisation en se basant sur des éléments autonomes et sur un relais du trafic entre les noeuds d'un même voisinage. Cette faculté va parfaitement s'adapter aux contraintes des technologies sans fil. De plus, l'étude menée par MANET tend à introduire les normes d'interconnexion de réseaux dans les contextes *ad hoc*. Avec l'introduction de TCP/IP, la topologie *ad hoc* semble la mieux armée pour permettre une extension facile, fiable et à faible coût d'un internet.

C'est pour ces raisons que pour offrir une ubiquité de connexion aux utilisateurs nomades à travers leurs unités mobiles, nous pensons que l'intégration des technologies sans fil va se faire en proposant des architectures hybrides comme le montre la figure 1.5. Dans ce type d'environnement envisagé, les architectures filaires composant Internet restent en place et sont étendues grâce à la mise en place de stations de base, comme c'est le cas actuellement par le biais des hotspot WiFi. Cependant, ces hotspot vont être considérés comme un élément autonome d'un réseau *ad hoc* et, ainsi, les unités mobiles pourront former des réseaux incluant les bornes d'accès qui serviront de routeur pour accéder à Internet. Ces topologies peuvent être construites dans les milieux urbains [HGB04].

Cependant en fonction des déplacements des unité mobiles, nous pouvons arriver à des situations, que nous nommons un *morcellement de réseaux* exposé par la figure 1.6, où un ensemble d'éléments peut se retrouver complètement isolé du reste d'internet. Notons qu'un élément ne restera jamais indéfiniment isolé. Dans de telles conditions, il paraît peu envisageable que le système soit encore en mesure d'assurer la gestion des communications distantes entre tout point d'un réseau, comme tel était le cas dans les réseaux filaires. Nous verrons que cela à une importance sur la façon de concevoir les applications distribuées (cf section 4.3).

Dans ce document, nous nous intéressons à la conception d'applications réparties sur ces environnements hybrides. Plus particulièrement, nous étudions le cas du morcellement de réseau où les unités ont une forte mobilité et où nous disposons uniquement d'informations sur le voisinage. Dans de tels environnements, nous verrons comment l'utilisation des agents mobiles va permettre de prendre en compte les problèmes liés à la mobilité physique des topologies hybrides. Mais, avant d'en arriver là, commençons par présenter les différentes méthodes de conception existantes pour les applications réparties avant de rentrer plus en détail dans celles basées sur les agents mobiles.

1.2 Les Applications Réparties

Après avoir exposé les différents types d'architectures réseau, nous allons présenter ici les paradigmes existant pour la conception d'applications réparties. Pour commencer, rappelons que les applications distribuées utilisent, au cours de leurs exécutions, un ensemble d'éléments, encore appelés *ressources*, qui peuvent être réparties sur différents sites du réseau sous-jacent. Pour accéder et utiliser les ressources, plusieurs schémas d'organisation sont envisageables. Pour cette section, nous nous basons sur la thèse [Rou02].

1.2.1 Les principaux schémas d'organisations

Nous pouvons référencer quatre grands types de schémas organisationnels. La conception d'une application répartie pouvant mélanger plusieurs de ces types, il s'agit ici d'une classification générale non stricte.

Mémoire distribuée partagée

Ce schéma d'organisation est basé sur la notion d'un espace d'échange partagé par un ensemble de processus distribués [BZS93]. Cette mémoire, généralement de grande taille, permet de faciliter les échanges de données pour les applications dont le partage est un facteur d'importance. Citons pour exemple Linda [Gel85] qui permet de mettre en place un mécanisme de tableau blanc distribué offrant les actions de dépôt, de retrait, de lecture et d'écriture d'objets.

Abonnement/publication

Ce schéma d'organisation permet de faire communiquer indirectement différents processus grâce à la collaboration d'un tiers. Le principe est de faire savoir, aux processus qui le souhaitent, qu'un évènement précis est arrivé dans l'environnement, la création d'un fichier par exemple. Pour réaliser ce type de communication, un gestionnaire d'évènements est présent au sein du système et les processus peuvent s'y abonner en fonction d'un certain nombre de critères. Le gestionnaire publie alors aux abonnés les évènements qui correspondent aux critères de souscription. Ce mécanisme est utilisé, par exemple, pour la détection de pannes dans un système informatique distribué. Si un site subi une avarie, le gestionnaire d'événements averti tous les autres éléments du système qui pourront alors adapter leur comportement ; par exemple, ils peuvent interdire toutes tentatives de communication avec le site en panne.

Pair à Pair

Dans ce schéma d'organisation, tous les éléments possèdent le même rôle au sein de l'application [Shi00]. Ces éléments sont appelés des *servants* qui possèdent un ensemble de voisins avec lesquels ils peuvent dialoguer en leur nom ou pour celui d'un autre. Ainsi, si deux servants non-voisins veulent communiquer, ils peuvent le faire grâce à un ensemble d'intermédiaires relayant la communication. Ce schéma, qui a été rendu populaire grâce aux échanges de fichiers sur internet [Dun01], est particulièrement adapté aux architectures dynamiques où il est difficile de mettre en place une gestion centralisée. Cette méthode de pair à pair ne se limite pas aux échanges de fichiers et peut être utilisée, par exemple, pour des applications coopératives avec une gestion plus simple de la mobilité des sites [PC02].

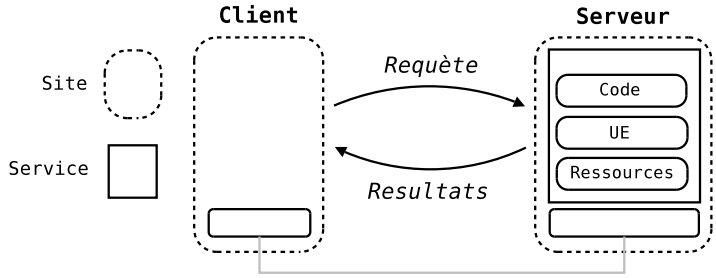


FIG. 1.7 – Schéma d'organisation Client/Serveur

Client/Serveur

Dans ce dernier schéma, on sépare une fonctionnalité précise de celui qui l'utilise grâce à une répartition des éléments et à une communication à distance. En d'autres termes, une application rendant une certaine fonctionnalité (un service) est placée sur un noeud du réseau (le serveur) en attente de requêtes émisses par d'autres applications (les clients) réparties sur le réseau. Ce schéma est en fait une manière de communiquer entre deux processus à l'aide d'un protocole préétabli. Lorsque qu'un client souhaite la réalisation d'un service précis, il envoie une requête au serveur qui exécute le service et lui renvoie le résultat. On retrouve ce mécanisme dans les applications Web où, par exemple, un navigateur (le client) récupère les pages d'un serveur Web (le service) afin de les afficher à l'utilisateur.

Le principal intérêt de ce type de schéma est qu'il peut être déployé indépendamment d'un élément tier, un composant système spécifique généralement, ce qui n'est pas le cas des schémas abonnement/publication et à mémoire distribuée partagée. De plus, avec une relation basée sur un protocole préétabli, le serveur garde un contrôle plus précis sur le déroulement des services. Enfin, les requêtes étant toujours à l'initiative du client, les serveurs n'ont pas besoin de vérifier l'état général du système et peuvent se concentrer sur la gestion propre du service.

Conclusion

Les différents schémas d'organisation que nous venons de présenter seront utilisés en fonction des besoins propres des applications à construire et surtout en fonction de contraintes liées au support physique (bande passante, hétérogénéité, etc.). Néanmoins, le plus populaire est sans conteste le modèle client/serveur grâce à un compromis entre la facilité de déploiement et la maîtrise du savoir faire lié à la fonctionnalité propre du service mais surtout grâce à la possibilité pour le client de sous-traiter des tâches sur des machines plus performantes.

En reprenant [Pic01], nous pouvons définir un service en trois composants élémentaires, présentés sur la figure 1.7, à savoir *le savoir faire* (le code), *les ressources nécessaires* (applicables au code) et *l'unité d'exécution* (qui traite le code). Pour permettre une efficacité accrue, il sera intéressant de tendre vers un regroupement des trois éléments sur le même site. Notons que, pour un client qui souhaite utiliser un service précis, il faut pouvoir identifier, trouver puis utiliser l'ensemble des trois éléments convoités. En conservant cette décomposition, nous allons à présent détailler toutes les possibilités de construction de ce schéma client/serveur, en nous focalisant sur la manière d'utiliser le service.

Pour cela, nous étudions l'évolution des paradigmes permettant la mise en œuvre d'un service que nous décrirons à travers les trois composants élémentaires et en regardant lequel ou

lesquels seront déplacés. Nous commençons par présenter les modèles d'exécution à distance, actuellement les plus répandus, où tous les éléments sont immobiles puis nous abordons des modèles utilisant la mobilité grâce aux déplacements d'un ou plusieurs composants d'un service.

1.2.2 Les différents types de requêtes

Nous présentons ici les différents mécanismes (protocoles) permettant aux clients d'envoyer leurs requêtes à un serveur. Pour cela, nous disposons d'une couche système prenant en charge la localisation des sites, le transport fiable et efficace des communications entre deux partenaires distants. Dans tous ces mécanismes, les trois éléments définis pour un service (code, ressource et UE) restent concentrés sur le serveur, la différence se faisant simplement sur la manière d'envoyer la requête.

L'envoi de messages

Cette méthode fût la première à être disponible lors de la construction des réseaux d'ordinateurs. Elle repose directement sur le service de communication offert par le réseau. Ce paradigme est réalisable à travers des bibliothèques de bas niveau généralement difficiles à mettre en œuvre [Rif98]. De plus, cette méthode n'est pas directement intégrée aux langages de programmation et nécessite de définir un protocole de communication stricte entre le serveur et ses clients. Ces deux contraintes ne permettent pas une programmation simplifiée aux développeurs. Un exemple d'application utilisant cette méthode est le serveur FTP (*File Transfert Protocole*) [Gie78].

Appel de procédure à distance

Dans la plupart des langages de programmation, on offre la possibilité aux développeurs de décomposer un code en plusieurs fonctions appelées suivant le déroulement de l'algorithme à réaliser. Cette méthode a été élargie en offrant la possibilité de réaliser des appels à distance grâce au mécanisme d'appel de procédure à distance ou RPC (*Remote Procedure Call*) [BN84, Nel81]. Ce mécanisme est une abstraction permettant aux développeurs de ne pas manipuler directement le service de communication réseau mais d'appeler une procédure distante comme si elle était locale. Ainsi, un processus utilisant le RPC ne fait aucune différence entre appel local et distant. Le service NFS (*Network File System*) [LSC+85] est construit sur ce mécanisme en permettant d'accéder à des fichiers distants comme si ils étaient sur le disque local d'une machine. Notons qu'il faut connaître à l'avance la procédure que l'on souhaite utiliser.

Appel de méthode à distance

Ce mécanisme se place à un niveau d'abstraction plus haut que celui des processus, à savoir la *programmation à objets* [Mey97]. Un objet est une entité logicielle regroupant en son sein des données et un comportement. Chaque objet réalise une fonctionnalité précise et est accessible à travers une interface qui regroupe un ensemble de méthodes[1]. Une application étant alors

[1]Cette ensemble peut contenir, plus rarement, des attributs

décrite comme un ensemble d'objets et d'interactions entre ces objets. Un objet est vu comme une boite noire capable d'effectuer une tâche définie et utilisable à travers son interface.

Pour permettre la distribution transparente des objets, on utilise généralement des intergiciels, comme CORBA [GGM99], qui vont s'occuper de rediriger les appels de méthodes à travers le réseaux et vont s'occuper de l'hétérogénéité des systèmes d'exploitation et/ou du matériel. Le mécanisme d'appel de méthode à distance est en fait une généralisation du mécanisme de RPC en se basant sur un *ORB* (*Object Request Broker*). À partir d'un identifiant précis et de la méthode que l'on souhaite exécuter, celui-ci s'occupe de la localisation des objets et de l'acheminement des appels afin de rendre totalement transparente la distribution aux concepteurs qui manipulent des objets répartis comme s'ils étaient locaux.

Invocation distante

Pour permettre de déployer des services directement sur Internet, les trois premières méthodes ne sont pas adaptées. Celle par message nécessite d'utiliser des bibliothèques de trop bas niveau difficiles à mettre en œuvre et les appels distants ont besoin d'un support réseaux sûr avec de faibles temps de réponse. Pour des raisons de sécurité, généralement seules les communications servant pour les serveurs Web sont accessibles sur Internet. Ces communications utilisent le protocole HTTP (*Hyper-Text Transfert Protocol*) qui va servir de base pour l'invocation de distante grâce à la norme SOAP (*Simple Object Acces Protocol)* [Ant04]. Celui-ci décrit les interactions entre les clients et un service Web à partir du langage à balise XML qui peut-être transmit directement par HTTP et évite l'utilisation d'un ORB.

Récemment, les web-services ont connus un réel succès car ils permettent de réaliser « des appels de méthodes » directement sur le Web, ce qui permet de favoriser, voire d'étendre, la collaboration entre une entreprise et ses clients, en externalisant ses services, tout en s'affranchissant des problèmes d'hétérogénéité matérielle et logicielle grâce à l'utilisation de langages de haut niveau.

1.2.3 Introduction de la mobilité

L'interaction de type client/serveur que nous venons de présenter se base sur des interactions distantes entre le client et le serveur. Nous présentons ici une autre manière de concevoir les interactions en introduisant la mobilité qui peut être supportée par une nouvelle famille de langages [Tho97]. Cette mobilité peut être vue comme une variation du modèle clients/serveur classique [CPV97]. Nous en présentons les différentes formes existantes en nous basant sur le déplacement d'un ou plusieurs éléments composant le schéma client/serveur (cf figure 1.7), *i.e* le code, les ressources ou l'unité d'exécution. Nous prenons comme référence le placement de chacun de ces éléments avant et après l'exécution du service [Pic98].

Envoi du savoir faire

Pour l'envoi du savoir faire, plus communément appelé *évaluation distante*, c'est le site du client qui dispose du savoir faire propre au service à réaliser. Les ressources et l'unité d'exécution étant au démarrage sur le serveur, la mise en route du service est réalisée après que le serveur ait reçu le code à exécuter. Les résultats sont renvoyés une fois le service achevé, le code et l'unité d'exécution sont alors supprimés. La figure 1.8 illustre ce mécanisme.

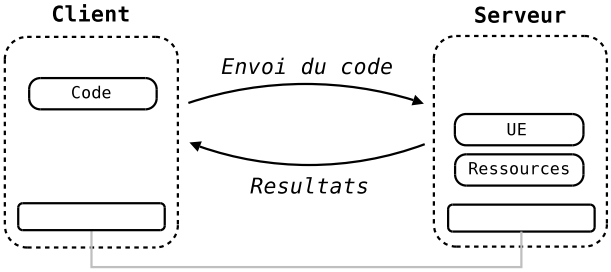


FIG. 1.8 – Envoi du savoir faire

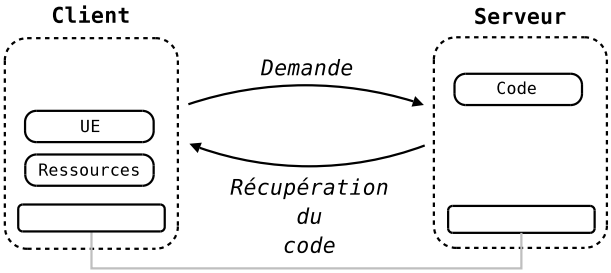


FIG. 1.9 – Récupération du savoir faire

Dans cette première utilisation de la mobilité, le client peut envoyer directement le code au serveur, comme dans le cas *rsh (Remote Shell)* qui permet d'exécuter un script[2] sur une machine distante, ou il peut utiliser un représentant local du service, comme pour l'impression d'un document sur une imprimante *Postscript* où un client s'adresse à un serveur local en lui communiquant un fichier, qui est ensuite envoyé à l'imprimante qui l'interprète dans une unité d'exécution propre. Nous pouvons aussi mentionner que ce schéma est réalisé par les requêtes *SQL*, adressées à un serveur de base de données, et aussi par les *servlets* Java déployés sur les serveurs HTTP munis d'un module approprié (J2EE, tomcat et autres).

Récupération du savoir faire

Dans ce cas, appelé aussi *code à la demande*, le client dispose de l'unité d'exécution et des ressources mais pas du savoir faire qui va être récupéré auprès du serveur. Il s'agit donc de l'inverse du cas précédent. Ainsi, un client adresse une requête uniquement pour récupérer un code précis afin de l'exécuter localement avec les ressources présentes. Les trois éléments sont réunis sur le site client comme le montre la figure 1.9. Cette méthode permet d'étendre les fonctionnalités d'une application directement chez le client sans avoir besoin d'effectuer une nouvelle installation. L'exemple le plus répandu d'utilisation de cette méthode est le téléchargement *d'applet* Java à partir d'un serveur Web. Ces applet sont des classes Java présentes sur le site serveur qui sont téléchargées puis interprétées par la machine virtuelle du site client.

Migration de processus

Ce schéma est utilisé lorsque l'on souhaite déplacer un savoir faire avec son unité d'exécution comme le montre la figure 1.10 [MDW99]. Ainsi, durant l'exécution du service, tous les

[2] un script est un fichier contenant une suite d'instructions déroulées par un interpréteur de commandes

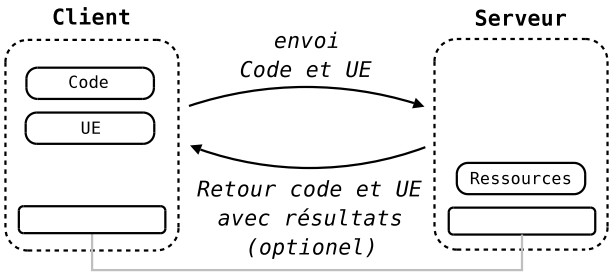


FIG. 1.10 – Migration de code

éléments se retrouvent sur le site serveur. L'intérêt de cette méthode est qu'elle permet d'appliquer le même code à des ressources différentes et réparties sur le réseau sans avoir à le télécharger auprès du client à chaque fois. On peut concevoir ce schéma comme une généralisation de l'envoi du savoir faire qui serait une succession d'envois vers une suite de serveurs.

Une application de ce genre de schéma est la réalisation d'une même requête *SQL* sur une suite de bases de données distinctes. Le processus est déplacé de serveur en serveur en fonction des ressources qu'il souhaite utiliser. Un autre exemple est l'équilibrage de charges sur un réseau de machines identiques, *i.e* un *cluster*. On va déplacer les processus afin que chaque site supporte la même charge de travail. Pour cela, si une machine est trop chargée, *i.e* qu'elle a plus de processus que les autres par exemple, on va déplacer un processus, à savoir son code et son unité d'exécution, vers une machine moins chargée.

Lorsque l'on déplace directement le code avec son unité d'exécution, on parle alors de *migration forte*. Celle-ci permet de continuer le flot d'exécution d'un programme comme s'il ne s'agissait que d'un simple passage par un état non-actif du processus. Notons de plus, que la migration de processus s'inscrit généralement dans un schéma où le déplacement est provoqué par le système, on parle alors de *migration réactive*. Le processus exprime ses besoins en ressources, comme l'accès à un fichier ou à du temps processeur, et c'est le système qui va placer le processus demandeur sur la machine correspondant le mieux à ses besoins. Par exemple, le placer sur la machine où le fichier est stocké ou équilibrer la charge. Nous pouvons dire que généralement la migration de processus est basée sur une migration forte et réactive. D'autres types de migration sont envisageables pour le même schéma de mobilité [FPV98], nous les verrons par la suite.

Agents mobiles

Le schéma de mobilité des agents mobiles dérive de deux domaines différents, *i.e* les agents venant de l'intelligence artificielle avec les systèmes multi-agents [Fer95] et des systèmes distribués avec la migration de processus [MDP+00]. Nous donnons plus de précision sur ces origines dans la section 1.3. Pour simplifier, nous pouvons dire que le schéma à agents mobiles est une généralisation de la migration de processus où le déplacement est à l'initiative même du code. Nous parlons alors d'une *migration proactive*

Les agents mobiles ont été introduits initialement en 1994 avec l'environnement Telescript [Whi94] qui permettait à des processus de choisir eux-mêmes de se déplacer sur les sites d'un réseau afin de travailler localement sur les ressources. Dans ce cas, la migration était forte et proactive. L'intérêt suscité par cette méthode fut confirmé avec le succès de Java et de son utilisation dans la programmation d'applications basées sur le Web. Malgré les qualités propres

de ce langage, la migration forte est encore, à l'heure actuelle, réalisable à titre expérimental [BHK+04] et n'est pas directement utilisable avec la version officielle. Ceci est dû principalement au fait que Java déplace des objets avec un état courant (par la sérialisation) et non pas un processus proprement dit ou un thread avec sa pile d'exécution.

Pour pallier à ce problème, plusieurs plates-formes ont proposé d'utiliser une migration dite *faible*, *i.e* où le code est déplacé avec un ensemble de données propres mais pas avec l'intégralité de l'unité d'exécution. En fait, dans une migration faible, l'agent est déplacé avec un état et est redémarré après chaque déplacement. Le développeur devra alors consulter l'état courant de son agent à chaque démarrage pour connaître l'avancement général de la tâche qu'il souhaite accomplir.

En conclusion, on retiendra qu'il existe différents types de migration de code [FPV98] qui peuvent être classifiés selon deux grandes caractéristiques : la première est la prise de décision de la migration (réactive/proactive) et la seconde est la capacité de déplacement du contexte d'exécution (forte/faible). Nous estimons que la caractéristique principale d'un agent mobile est de disposer d'une certaine autonomie grâce à la migration proactive, qu'elle soit forte ou faible. C'est cette autonomie qui différencie les agents mobiles de la migration de processus.

1.2.4 Conclusion : des applications multi-agents mobiles

Dans cette partie, nous avons vu différentes méthodes utilisables pour la construction d'applications réparties. Pour commencer, nous avons présenté les différents schémas d'organisation de services que sont la mémoire distribuée partagée, l'abonnement/publication, le pair à pair et le schéma client/serveur. Le choix du schéma s'opére principalement en fonction de contraintes physiques propres aux réseaux etudiés au début de ce chapitre. Par exemple, la mémoire distribuée partagée pourra convenir à un réseaux de hautes performances mais sera peu adaptée aux larges échelles comme celle d'Internet. En règle générale, le schéma le plus répandu reste le client/serveur qui permet de séparer correctement la fonctionnalité propre d'un service de celui qui va l'utiliser.

Le client/serveur permet de mettre en place des interactions qui ne nécessitent pas de service système autre que la communication distante proposée par la couche réseau. Celle-ci prend en charge la localisation des sites, le transport fiable et efficace des communications entre deux partenaires distants. À partir de cela, nous avons présenté différentes méthodes d'interaction permettant aux clients d'effectuer les requêtes de service auprès des serveurs. Ces interactions sont réalisables à distance grâce aux hypothèses que nous avons faites sur le réseau.

Ensuite, nous avons étudié une variation du schéma client/serveur en introduisant la mobilité d'un ou plusieurs éléments le composant. Nous avons vu que cette mobilité peut prendre plusieurs formes : l'envoi de savoir faire, la réception du savoir faire, la migration de processus et enfin les agents mobiles. Selon les éléments déplacés, chacune de ces formes apporte des solutions à certains types d'applications distribuées. Avec les architectures hybrides que nous avons présentées (cf. 1.1.3), nous pensons que les qualités propres des agents mobiles seront les plus concordantes avec les environnements à venir, surtout avec l'utilisation des technologies sans fil [BK04]. Les avantages et inconvénients des agents mobiles sont le sujet de la dernière section de ce chapitre.

Sans rentrer dans les détails de la conception des applications à base d'agents mobiles, qui est l'objet du chapitre 4.3, nous pouvons dire qu'un agent réalise une fonctionnalité précise et relativement simple. Qu'elle soit cliente ou serveur, une application réalisant une tâche

plus complexe sera construite grâce à plusieurs agents, comme dans le cadre de la conception orientée objet où une application est un ensemble d'objets et de relations. Nous pouvons donc abstraire les applications réparties comme un ensemble d'agents, pouvant se restreindre à un singleton, se déplaçant de sites en sites et dialoguant entre eux. Ces agents se déplaceront pour deux raisons principales : trouver et utiliser les services applicatifs désirés, comme un serveur de fichiers, ou pour mieux s'adapter à leur environnement, par exemple migrer lorsqu'un site est trop chargé. Les phases de dialogue servent, quant à elles, à mettre en commun des fonctionnalités ou simplement à connaître l'avancement de la tâche générale.

1.3 Étude du concept d'agent

Avant de passer à l'étude plus approfondie des agents mobiles qui constitue le prochain chapitre, nous présentons un certain nombre de points servant à préciser le cadre de notre étude. Nous commençons par donner une définition plus rigoureuse des agents mobiles en comparaison avec les différents domaines utilisant le terme agent. Ensuite, nous regardons comment l'autonomie des agents peut avoir une influence sur leur conception. Enfin, nous regardons comment la tâche générale d'un agent mobile peut être décrite à partir d'une suite de comportements et nous terminons par une étude des différents formalismes de description pour les agents.

1.3.1 Les différents types d'agents

Le terme d'agent est issu principalement de deux domaines distincts : les systèmes multi-agents et les systèmes distribués avec la migration de processus. Les systèmes multi-agents appartiennent, à la base, au domaine de l'intelligence artificielle où la programmation tente d'imiter l'esprit humain lors de l'exécution [AS92]. Dans ce cadre, on désigne par *agent cognitif*, un programme qui possède une « intelligence » assez développée pour prendre des décisions face à des situations complexes. Dans ce contexte, l'agent est isolé, *i.e* il n'a aucune interaction avec tout autre agent, et il se contente d'itérer seul son algorithme afin de résoudre le problème qui lui est posé.

Les difficultés rencontrées par les agents cognitifs pour résoudre des problèmes complexes et distribués, à cause de leur approche centralisée principalement, ont poussé à repenser leur fonctionnement en distribuant l'intelligence dans différents éléments communiquant entre eux [San04]. C'est de cette idée d'où viennent les systèmes multi-agents. Les agents sont alors définis comme des entités, physiques ou virtuelles, pouvant agir volontairement sur leur environnement et communiquant entre elles [Fer95]. Un agent peut donc être un robot qui agit sur son environnement en déplaçant un objet ou simplement un capteur de mouvement qui communique avec une unité de surveillance.

D'un autre côté, l'agent système est généralement un programme autonome s'exécutant au sein d'un système distribué « multi-agents », comme dans la norme MASIF [MBB+98]. Il n'a pas vocation à résoudre des problèmes complexes et se limite généralement à l'exécution de tâches triviales. On peut dire, par exemple, qu'un serveur Web est un agent. Il est lui aussi capable de prendre des décisions par rapport à ses ressources, mais cette capacité de choix reste restreinte. Ici, les agents n'agissent pas intentionnellement sur l'environnement et peuvent communiquer entre eux grâce à un médium de communication géré par le système.

À partir de ces différentes approches, nous définissons donc un agent comme un élément autonome possédant une activité interne avec ses propres ressources, œuvrant généralement au nom d'un utilisateur ou d'une application, communiquant avec d'autres agents afin de réaliser la tâche pour laquelle il a été créé [MDW99]. Les agents mobiles sont alors définis comme des agents ayant la capacité de se déplacer de site en site et ayant la conscience de leurs déplacements.

1.3.2 Les comportements et les transitions

Nous avons vu dans la description des différents types de mobilité que la migration forte permet de déplacer un processus, ou un agent, avec ses données et son unité d'exécution. Ainsi, après un déplacement, l'exécution poursuit son cours comme si elle ne s'était pas arrêtée. Cependant, ce type de migration, qui se base sur une capture implicite de l'état d'exécution d'un agent, est difficile et coûteuse à réaliser surtout dans les environnements hétérogènes où vont évoluer les agents. Comment faire correspondre la capture d'un état, basé généralement sur l'unité d'exécution, dans deux interpréteurs différents ? En plus, d'un point de vue conceptuel, nous avons dit qu'un agent a conscience de ses déplacements et que généralement il effectue une migration proactive, *i.e* volontaire. Dans de telles conditions, c'est le concepteur qui va choisir de faire déplacer son agent. Il est donc capable de déterminer les éléments qu'il souhaite conserver après son déplacement et surtout quelle tâche l'agent va entreprendre lors de sa prochaine visite.

Plusieurs études se sont portées sur la possibilité de définir les différents aspects que l'on peut retrouver dans un agent. [KKPS98] propose de découper un agent en quatre grandes couches : mobilité, adaptation, action, collaboration. Cette première approche permet de séparer les grands aspects d'un agent en montrant que le concepteur peut se focaliser sur certaines couches et laisser la gestion des autres au système. Dans une autre étude, [LJJM01] propose de se détacher du modèle classique de programmation, où un code déroule une suite d'instructions parmi lesquelles figurent les migrations, vers un modèle décrivant un agent en trois aspects différents : fonctionnel, migratoire et management. Cette description permet au concepteur de s'occuper principalement des fonctionnalités des agents en dérivant le passage d'une fonction à une autre et laisse la partie système gérer la migration et la mise en place des fonctionnalités. Ceci permet d'exécuter l'agent dans des environnements hétérogènes.

Nous allons donc reprendre le concept permettant de séparer l'aspect fonctionnel d'un agent de celui de la migration. Ainsi, on peut utiliser la description de la tâche générale d'un agent mobile comme un ensemble de sous-tâches à réaliser entre chaque migration. Nous définissons alors un agent comme un ensemble de comportements (fonctionnalité locale) et de transitions entre ses comportements. Une migration peut être vue comme une transition avec un changement de localisation.

1.3.3 L'autonomie

Nous verrons plus en détails dans la prochaine section (1.4) concernant les avantages et inconvénients, que les agents mobiles ont un avantage principal sur le client/serveur grâce à leur capacité d'adaptation dans les environnements dynamiques hétérogènes. Cette faculté est essentiellement due à leur aptitude au déplacement en fonction de leurs besoins propres pour accomplir au mieux la tâche qui leur incombe.

D'un autre côté, nous avons vu que nous pouvons avoir deux grandes classes de migration, *i.e* la migration proactive ou la migration réactive. Avec la migration proactive, c'est l'agent qui initie le déplacement, ceci permettant de garder intégralement leur caractère autonome. Mais avec la migration réactive, c'est le système qui initie les déplacements sans avoir besoin d'une demande explicite de l'agent. Cette migration réactive pose un problème du point de vue de l'autonomie. Comment la garantir si l'agent peut être déplacé sans son accord et si l'agent est déplacé sur un site A par le système alors qu'il souhaitait se rendre sur un site B ?

Cependant, la migration réactive intervient généralement lorsque l'on souhaite mettre en place des migrations totalement transparentes aux unités déplacées. Dans ce contexte, chaque unité retrouve, avant et après la migration, le même environnement et n'a absolument pas conscience de son déplacement. On peut prendre l'exemple des clusters où les processus sont déplacés et ont toujours accès aux mêmes fichiers. Par contre, l'adaptation d'un agent est nécessaire lorsque l'environnement change et qu'il ne peut plus trouver les ressources dont il a besoin. Dans le cas où il peut toujours trouver ce dont il a besoin, cela ne nuit pas à son autonomie.

Ainsi nous définissons la migration des agents mobiles comme proactive afin de garder au maximum leur caractère autonome. Cependant, lorsqu'un agent arrive dans un environnement où le déplacement s'effectue sans changement de contexte, il acceptera de se déplacer selon les ordres du système. En d'autres termes, la migration d'un agent est par défaut proactive, mais lorsqu'il arrive dans un environnement homogène, tel un cluster, il accepte volontairement de subir des migrations réactives sous garantie de toujours avoir accès aux ressources qu'il exploite.

1.3.4 Formalisme de description

Les agents, et en particulier, les agents mobiles, constituent un concept utilisé dans différents domaines et dont les contours sémantiques sont assez larges. Il est donc difficile d'en trouver une formalisation unique. Si l'on se limite aux agents mobiles et communicants, leur formalisation a fait appel d'une part aux systèmes de transitions, avec Mobile Unity [MR98] par exemple, d'autre part aux algèbres de processus avec comme base le π-calcul.

Des extensions du π-calcul ont été proposées, dans le cadre du projet SARDES par exemple, pour traduire la mobilité des agents [SS04]. De même, une architecture de communication entre agents mobiles est formalisée dans [SWP99] en utilisant une extension du π-calcul. Du point de vue de la mobilité, on peut aussi raisonner sur la localisation des agents. Dans le cas de la logique Klaim [BLD02] par exemple, les propriétés des agents sont exprimées dynamiquement en fonction de l'évolution de leur localisation. Ces recherches constituent encore des études très prospectives et leur utilisation est, en elle-même, un sujet de recherche.

Nous avons vu dans la conclusion de la section précédente (1.2.4) que les applications à base d'agents mobiles peuvent être conçues de la même manière que les applications orientées objet. Dans ce cadre, il existe des méthodes semi-formelles, telles que UML, qui permettent de décrire la structure des applications en explicitant les interactions entre objets [Obj03]. Ces techniques de conception ont été adaptées au domaine des agents afin de formaliser les interactions, comme dans AUML [PRT+03]. Cependant ces techniques viennent du domaine des systèmes multi-agents qui se concentrent plus sur l'expression des relations inter-agents. L'expression de la mobilité du point de vue des systèmes distribués n'est pas décrite.

1.4 Avantages et inconvénients des agents mobiles

Les agents mobiles ont rapidement suscité un intérêt tout particulier dans les domaines de recherche portant sur les applications réparties. Très rapidement, cette nouvelle méthode de programmation a été évaluée afin de voir ce qu'elle pouvait apporter comme caractéristiques propres [CHK94] et ce qu'elle permettait d'améliorer par rapport aux méthodes de programmation plus classiques [Ber99b]. Dans cette section, nous allons exposer ces différents apports tout autant que les difficultés soulevées par les agents mobiles.

1.4.1 La performance

Les premières améliorations apportées par les agents mobiles portent sur le gain de performances due à une meilleure utilisation des ressources physiques mises à disposition. L'amélioration va intervenir à différents niveaux permettant d'optimiser la tâche globale des agents.

Diminution de l'utilisation du réseau

Le déplacement des agents mobiles permet de réduire significativement, voir supprimer, les communications distantes entre les clients et les serveurs. En privilégiant les interactions locales, l'utilisation du réseau va se limiter principalement au transfert des agents. Cette situation présente trois principaux avantages.

Le premier avantage est la diminution de la consommation de bande passante. En effet, plusieurs études [SM98, GCK+02, Ber99a] montrent qu'en comparaison de l'envoi à distance de requête (procédures et méthodes), la mise en place des agents mobiles permet d'obtenir une réduction significative de la charge réseau en terme du nombre total de données tranférées. Cette diminution est constatée dans différents types d'applications nécessitant d'intenses échanges d'informations entre le client et le serveur. Nous pouvons donner à titre d'exemple, la collecte d'informations dans des bases de données réparties, l'exploration d'un internet ou encore la gestion de réseaux.

Le second avantage notable est la diminution des temps de latence [Joh98, KG99, JXHX02]. Dans le contexte des réseaux à large échelle, la mise en place d'applications reparties, nécessitant de fréquentes interactions entre client et serveur, se heurte aux temps de latence propres aux communications réseaux. Il arrive fréquemment que le temps d'attente de la réponse d'une requête soit plus long que le temps de traitement nécessaire à la réalisation du service. En rapprochant client et serveur dans un même sous-réseau, voire sur un même site, on les place dans un environnement où les temps de réponse des interactions sont limités, ce qui permet de réduire d'autant les temps de latence.

Le dernier avantage à souligner vient des brèves périodes de communication. En réduisant le plus possible les communications distantes aux seuls transferts d'agents mobiles, on diminue considérablement les périodes de connexion entre deux sites. Cette diminution de la fenêtre d'utilisation des communications réseaux permet de moins se soucier des ruptures de liens physiques qui peuvent intervenir fréquemment dans les environnements sans fil.

Des calculs indépendants

Dans le modèle classique d'évaluation à distance, le client et le serveur doivent rester connectés tant que le service est en cours d'exécution. En plus des problèmes liés aux fluctuations des performances réseaux, certains services, nécessitant de longues phases de traitement, ne supportent pas facilement une rupture de connexion avec le client. Dans ce cas, ils doivent souvent redémarrer entièrement leurs calculs. Mais, le maintien du lien de communication peut s'avérer difficile dans des réseaux à large échelle ou sans fil. Avec les agents mobiles, un client peut déléguer les interactions avec le service sans maintenir une connexion de bout en bout [GCKR01]. Avec cette possibilité d'un calcul indépendant, le client peut demander un service, se déplacer (ou simplement terminer une session) puis venir récupérer les résultats plus tard. Ce mode de fonctionnement est particulièrement intéressant lors du lancement à distance de simulations numériques.

Optimisation du traitement

L'optimisation des phases de traitement se produit à deux niveaux. Premièrement, comme nous l'avons déjà dit, en localisant les ressources et le savoir faire sur un même site, on supprime les phases de dialogue entre le client et le serveur qui sont perturbées par des temps de latence dus aux communications réseaux [IH99]. Ensuite, le déplacement du savoir faire va permettre de déléguer les calculs sur des machines serveurs, un super-calculateur par exemple, qui sont généralement plus puissantes qu'une machine cliente. Cela est particulièrement vrai dans l'informatique nomade où la miniaturisation s'accompagne d'une perte de puissance significative.

Tolérance aux fautes physiques

En se déplaçant avec leur code et données propres, les agents mobiles peuvent s'adapter facilement aux erreurs systèmes. Ces erreurs peuvent être d'ordre purement physique, disparition d'un noeud par exemple, ou d'ordre plus fonctionnel, arrêt d'un service par exemple. Si on prend le cas d'un site perdant une partie de ses fonctionnalités, un service tombant en panne, l'agent pourra alors choisir de se déplacer vers un autre site contenant la fonctionnalité désirée. Ceci permet une bien meilleure tolérance aux fautes que le modèle statique classique.

Cependant, notons que dans le cas d'un agent unique, s'il vient à disparaître pour une erreur interne à son savoir faire ou pour une erreur système, il est généralement très difficile de détecter sa disparition. [Rou02] énumère plusieurs mécanismes basés sur la réplication, transparents aux agents, permettant de résoudre en partie ce problème.

1.4.2 La conception

Du point de vue de la conception, les agents mobiles représentent à la fois une méthode permettant de mieux caractériser certaines applications mais ils apportent aussi une complexité accrue par rapport au classique client/serveur bien mieux maîtrisé. Pour commencer, en règle générale, les concepteurs préfèrent avoir une méthode permettant de décrire facilement un comportement réel [MDW99]. Avec la méthode classique, il va être très contraignant de décrire des algorithmes d'exploration (de réseau) ou bien encore de caractériser les déplacements des utilisateurs nomades. Avec les agents mobiles, les concepteurs disposent d'une méthode qui

permet de décrire naturellement ce genre de comportement. Ainsi, on peut facilement mettre en place un déploiement et/ou une maintenance d'application sur un réseau [Pic98] ou encore suivre les utilisateurs dans leurs déplacements [Sat02].

Ensuite, les agents possèdent une capacité de traitement spécifique leur permettant de s'adapter à leur environnement. Généralement, dans le modèle client/serveur, le service est caractérisé préalablement par une interface stricte et/ou avec un protocole d'utilisation bien défini. Ainsi, si le client n'a pas connaissance de la description du service, il ne pourra pas l'utiliser. Par contre, les agents pourront eux s'adapter aux caractéristiques des services afin d'exprimer leur requête. Par exemple, si un service demande une communication sécurisée, l'agent pourra récupérer un module de sécurité, mettre à jour sa pile de communication et dialoguer avec le service [ITK99]. On peut aussi tout à fait imaginer que l'agent pourra s'adapter aux conditions du réseau en se séparant d'une partie de ses fonctionnalités pour s'adapter aux terminaux pauvres en ressources [LK02].

La capacité de raisonnement va permettre de concevoir des agents qui seront autonomes, adaptant leurs déplacements en fonction de l'environnement et pouvant moduler leurs fonctionnalités en cours d'exécution. Cependant ces propriétés s'accompagnent d'une conception bien plus difficile [Vig04] que celle du classique client/serveur. En effet, dans la majorité des applications distribuées, on essaie le plus possible de cacher la répartition en s'appuyant sur un ensemble de services système qui permettent de concevoir une application comme si tous ses éléments étaient locaux. C'est le cas de CORBA par exemple [GGM99]. Pour pleinement tirer partie des agents mobiles, la distribution doit être explicite et gérée par les agents eux-mêmes, compliquant d'autant la tâche des concepteurs. De plus, si on se réfère à la conception objet où une application est définie comme un ensemble d'éléments et de relations, il est difficile de bien définir la tâche de chaque agent (élément) et surtout de savoir comment et où les interactions (relations) vont s'opérer. Enfin, la complexité même des services rendus par les applications impose souvent aux concepteurs d'utiliser une méthode parfaitement maîtrisée plutôt que de recourir à un mécanisme dont ils n'ont pas vraiment l'habitude.

1.4.3 Le développement

Le domaine des agents mobiles étant encore relativement jeune, ils se heurte à de fortes contraintes lors des phases de développement. La première d'entre elles est qu'il existe à l'heure actuelle un trop grand nombre d'intergiciels pour agents mobiles qui possèdent chacun leurs propres défauts et qualités [Joh04]. Il suffit pour s'en convaincre de regarder le site de la « *Mobile List* » [MAL] qui référence toutes les plates-formes connues. Avec cette offre pléthorique, il est difficile de parler de standardisation et aucune ne semble encore s'imposer. En sachant que les agents doivent s'adapter aux conditions de l'environnement, les développeurs sont confrontés à un éventail de possibilités bien trop large. Nous étudions plus précisément un certain nombre de ces intergiciels dans le chapitre 2.

Le manque de standardisation se retrouve aussi dans les interactions entre agents, ainsi il n'existe pas de langage partagé par toutes les plates-formes. Ceci représentent un handicap sérieux, car les agents ont besoin d'exprimer les caractéristiques des services qu'ils recherchent et surtout d'obtenir des réponses précises sur leur localisation ainsi que sur la manière de les utiliser. De plus, avec tous les langages existants, les développeurs sont, encore une fois, face à un trop large éventail de possibilités. Pour résoudre ce problème, il faudra se tourner vers le domaine des systèmes multi-agents qui cherche à mettre un place des langages précis en

s'appuyant sur un ensemble d'ontologies et de protocoles caractérisant les interactions inter-agents [ARB03].

Le second problème important lors des phases de développement est la complexité de mise au point des programmes qui constitue une partie critique du processus de développement. La plupart des environnements de programmation possèdent des outils permettant de suivre les différents éléments d'une application durant son exécution afin d'en trouver les erreurs. Cependant, ce genre d'outil s'utilise parfaitement avec des éléments statiques mais est difficilement applicable dès le moment où les éléments se déplacent. Ceci est particulièrement vrai dans les architectures hybrides et/ou à grande échelle où il est quasiment impossible de surveiller l'intégralité d'un système. Ainsi, il peut s'avérer impossible de récupérer l'état d'erreur d'un élément se trouvant sur un site déconnecté. Sans outil de déboguage, il devient très difficile de différencier erreurs de conception et de développement. Notons que des efforts pour proposer une premier outil de débuggage est réalisé par la plate-forme JADE [BPR99].

Le troisième problème important, en relation avec le déboguage, est la difficulté de mettre en place une vérification des applications à base d'agents mobiles. La technique la plus utilisée encore de nos jours, bien qu'elle ne soit pas totalement satisfaisante, est sans aucun doute le test. Dans cette méthode, on va mettre à l'épreuve une application en lui injectant des données dans ses différents points d'entrée dans le but de simuler le comportement utilisateur et de recouvrir une plage de situations la plus large possible. Si le test s'applique assez naturellement dans des programmes classiques, *i.e* non répartis, il représente un domaine de recherche à part entière dès qu'il s'agit des applications répartie [BB02, UK99]. Dans ce cadre, il faut être capable de coordonner les différentes injections sur les points d'entrée répartis afin de garantir que le comportement simulé est bien celui qui était visé. Cette méthode est déjà loin d'être simple dans le cas d'une application répartie statique, on peut alors facilement imaginer que cela devient encore plus compliqué si les points d'entrées (les agents) deviennent mobiles. Pour vérifier les applications construites sur des agents mobiles, il faut utiliser une méthode permettant de garantir le comportement et les interactions des agents avant leur déploiement.

L'analyse statique de programme permet de déterminer statiquement, sans exécuter un programme, des propriétés qui seront satisfaites lors de l'exécution [Col97]. Cette méthode, qui permet de garantir un certain niveau de correction du programme analysé, semble toute indiquée pour permettre de se passer des phases de test et, ce, même dans le cas des applications réparties [PS93, KNY95]. Notons tout de même que l'analyse statique à fait ses preuves dans le cadre de la programmation classique mais qu'elle constitue toujours un domaine de recherche pour les applications réparties.

1.4.4 La sécurité : le talon d'Achille des agents mobiles

Le dernier point que nous abordons dans cette section porte sur le thème de la sécurité. D'un point de vue logiciel, la sécurité [RG91] consiste à empêcher des accès, et/ou des modifications, non-autorisés aux éléments d'un système informatique. Dans ce cadre nous faisons la distinction entre les *demandeurs* (utilisateur, processus, périphérique ...) et les *éléments demandés* (processeur, fichier, mémoire ...). Lorsque l'on souhaite avoir un système sûr, on met en place une politique de sécurité qui doit garantir la *confidentialité*, *i.e* les données des éléments ne sont pas divulguées aux demandeurs non-autorisés, et *l'intégrité*, *i.e* les éléments ne sont pas modifiés par des demandeurs non-autorisés [CCD+02]. Nous omettons volontairement la disponibilité qui doit garantir, aux demandeurs autorisés, l'accessibilité aux éléments et qui est plus du ressort de la tolérance aux fautes.

Pour mettre en place la politique de sécurité visée, on utilise généralement des mécanismes d'authentification (mot de passe, certificat ...), de cryptographie (SSH, SSL ...) et de contrôle d'accès (droit utilisateur, pare-feu). L'authentification à pour but d'identifier précisément le demandeur, la cryptographie doit assurer la confidentialité des données échangées et le contrôle d'accès vérifie l'adéquation entre demandeur et éléments demandés.

Dans le schéma classique des applications réparties, on regroupe les éléments critiques sur des machines sécurisées et on se focalise sur les canaux de communication externes en utilisant les mécanismes d'authentification et de cryptographie. Pour la mise en place de la mobilité de code, les politiques de sécurité se basent généralement sur une relation étroite entre le site stockant le programme et celui qui l'exécute [Zac03]. Implicitement, le possesseur du code est le même que celui de l'environnement qui va l'exécuter.

Pour ce qui est des agents mobiles, d'un point de vue de la sécurité, nous sommes face à un problème qui n'est pas encore intégralement résolu. C'est d'ailleurs le principal argument avancé pour expliquer la faible utilisation de ce paradigme [Rot04, Vig04]. En effet, les agents mobiles représentent un nouveau champ d'investigation pour le domaine de recherche en sécurité, d'une part dans la protection des sites vis-à-vis des *agents malveillants* et d'autre part dans la protection des agents vis-à-vis des *sites malveillants*. Nous présentons ces deux champs en nous basant sur la thèse de [Lou01].

Protection des sites

La protection des sites contre des attaques menées pas des agents malveillants est un problème qui est aujourd'hui bien maîtrisé. En effet, plusieurs solutions permettent maintenant de se prémunir d'éventuelles attaques et voici les méthodes les plus connues :

Bac à Sable La technique du bac à sable consiste à exécuter un agent à l'intérieur d'un environnement restreint, en interdisant l'accès au système de fichiers par exemple. Ainsi, un site peut exécuter un agent douteux dans le bac à sable sans trop se soucier des problèmes de sécurité. Cette approche peut facilement se mettre en place en utilisant des interpréteurs de code dont leurs possibilités sont limitées. Pour illustrer cette technique nous pouvons citer les *applet* Java exécutés à l'intérieur d'un navigateur Web.

Signature de code La signature de code intervient lors de la création d'un agent, son créateur le signant numériquement afin qu'il puisse s'identifier durant ses déplacements. Cette technique permet d'obtenir une authentification de haut niveau pour les sites. Les *applet* Java adoptent désormais ce mode de fonctionnement. Ainsi si une *applet* est signée, elle est considérée comme un code de confiance et peut accéder à toutes les fonctionnalité de Java. Elle sera placé dans un bac à sable dans le cas contraire.

Contrôle d'accès Pour améliorer les deux précédentes techniques, on met en place une politique de contrôle d'accès plus complexe. On peut la voir comme un raffinement d'une politique de bac à sable générale vers une politique spécifique à chaque application ou classe d'agents. En fonction des agents, le site pourra autoriser l'accès à un ensemble précis de fonctionnalités. Le contrôle d'accès permet de mixer les deux premières

techniques en offrant aux agents signés plus de fonctionnalité qu'un simple bac à sable sans pour autant accéder à toutes les fonctionnalités.

Vérification du code La vérification de code permet d'obtenir une garantie sur la sémantique d'un code à travers l'analyse de sa structure, ou de son comportement pour un agent, en fonction d'une politique de sécurité donnée. Les bacs à sable font déjà des vérifications rudimentaires en cours d'exécution, principalement pour garantir le typage des opérandes des instructions, mais cela est fortement coûteux. Une autre approche est de vérifier automatiquement le code, avant son lancement, en s'appuyant sur une preuve de conformité. Pour cela on peut utiliser l'approche *PCC (Proof Carrying Code)* [NL98]. Lors de la mise en route de l'agent, son créateur fournit un ensemble de preuves intégrées qui est transporté par l'agent. Ces preuves garantissant le comportement de l'agent en fonction de critères de sécurité des sites à visiter. Lorsque l'agent commence une nouvelle visite, le site récupère la preuve lui correspondant et vérifie si elle correspond à sa politique de sécurité. Le site choisit alors d'exécuter ou non l'agent. Cette technique se base pour l'instant sur des propriétés de typage et la preuve est fournie par le compilateur.

Protection des agents

À l'opposé de la protection des sites, la protection des agents contre des sites malveillants ne dispose pas de solution éprouvée et reste encore aujourd'hui un champ de recherche ouvert. Pour comprendre ce que risque un agent lors de son exécution sur un site malveillant, nous pouvons référencer les éléments transportés pouvant être cible d'attaque [Lou01] :

- ***Le Code*** : ensemble des instructions composant la tâche de l'agent.
- ***Les données statiques*** : données ne changeant pas durant les déplacements (la signature par exemple)
- ***Les données collectées*** : ensemble des résultats obtenus au cours des déplacements réalisés par l'agent depuis son lancement.
- ***L'état courant*** : ensemble de données servant à l'exécution courante de l'agent.

La sécurité des agents mobiles consiste alors à garantir les critères de confidentialité et d'intégrité de l'ensemble de ces éléments. Du point de vue des données, il est évident qu'un agent ne souhaite pas divulguer des informations critiques à n'importe quel site. Par exemple, un site malveillant pourrait récupérer la signature d'un code et l'utiliser pour créer un nouvel agent afin de s'introduire dans des environnements auxquels il n'a normalement pas accès. Pour le code, un agent transporte un savoir-faire propre à son concepteur qui pourrait tomber aux mains de ses concurrents.

Nous pouvons classifier trois grandes catégories d'attaques que les sites sont capables de réaliser : l'inspection, la modification et le rejeu [Zac03]. L'*inspection* consiste à examiner le contenu de l'agent, ou le flot d'exécution afin de récupérer des informations critiques transportées par l'agent. La *modification* se réalise en remplaçant certains éléments de l'agent dans le but de conduire une attaque. En remplaçant le code, l'agent effectuera des opérations malveillantes

sur les futurs sites à visiter. Le *rejeu* s'obtient en clonant l'agent puis en exécutant le clone dans plusieurs configurations pour retrouver le savoir de l'agent. Différentes techniques sont en cours d'étude pour garantir aux agents qu'ils peuvent accéder à des nœuds dans lesquels ils peuvent avoir toute confiance ou pour détecter les agents corrompus. Cependant, aucune d'elle n'apporte un niveau de sécurité suffisant comme celui proposé pour protéger les sites.

Ce que nous pouvons dire pour conclure sur les problèmes de sécurité liés à l'utilisation des agents mobiles, c'est qu'il s'agit encore d'un champ d'investigation complet. Bien que la protection des sites soit à présent maîtrisée, la protection des agents n'est, pour l'instant, toujours pas satisfaisante. Les différents efforts menés par la communauté de la sécurité nous poussent à croire que nous obtiendrons le niveau de sécurité requis [GCKR98, TBV04].

1.4.5 Bilan des avantages et inconvénients

À travers les différentes caractéristiques des agents mobiles que nous venons de présenter, nous pouvons dire que les agents mobiles ne représentent pas le paradigme idéal pour tous les types d'applications réparties, mais simplement une possibilité nouvelle pour la construction des applications. Selon les besoins de performance, de conception, de développement et de sécurité les agents mobiles pourront apporter une alternative intéressante au classique client/serveur dans certains types de configuration.

Le critère de performance des agents sera très utile dès que les applications vont comporter des communications intensives. Grâce à leurs interactions locales, les agents permettent de ne pas subir les ralentissements dûs aux bandes passantes limitées et aux temps de latence des réseaux, surtout sur Internet. De plus, en utilisant les machines serveurs, généralement plus performantes que les clients, les phases de traitement seront plus rapides. D'un point de vue conception, les agents mobiles permettent de décrire des comportements plus difficiles à caractériser avec le client/serveur. Ainsi, l'exploration d'un réseau, la représentation d'un utilisateur, le support des ruptures de communication ou encore l'adaptation à l'environnement sont naturellement exprimables grâce aux agents mobiles.

Cependant, ces qualités s'accompagnent d'importantes difficultés de développement. Elles sont dues principalement au manque de standardisation des intergiciels qui ne permettent pas d'obtenir, par exemple, un langage homogène partagé par tous les agents. On peut ajouter à cela, les difficultés de mise au point dues aux déplacements de l'unité d'exécution qui sont difficiles à suivre et aux problèmes de test qui nécessitent d'importants efforts de coordination. Ces difficultés de développement poussent les concepteurs à se tourner vers des paradigmes bien mieux maîtrisés en mettant en place des mécanismes de mobilité plus limités. D'un point de vue plus informel, la programmation classique dispose d'outils graphiques d'aide au développement qui intègrent complètement les phases de programmation, de déboguage et de test. Quelques outils du même type apparaissent pour les agents mobiles mais ne sont pas complètement intégrés en restant pour la plupart focalisés sur la phase de programmation [TOH01] et seuls quelques-uns s'intéressent à la phase de déboguage [BCTR05].

Pour terminer cette partie, nous pouvons dire que le problème le plus important empêchant l'adoption actuelle des agents mobiles est sans nul doute la sécurité. En effet, même si la protection des sites est quasiment assurée, celle des agents reste un réel problème qui n'a pas de solution définitive. Différentes études sont actuellement menées pour permettre d'obtenir un niveau de sécurité satisfaisant. Dans tous les cas, en gardant à l'esprit qu'il s'agit d'un

problème crucial, nous allons considérer pour le reste de notre étude que nous disposons d'un environnement sûr, composé uniquement d'agents et de sites de confiance.

1.5 Conclusion

Dans ce chapitre, nous nous sommes intéressés à décrire le contexte général où se situe notre étude sur les agents mobiles coopérants. Nous avons commencé par présenter les différentes architectures constituant un internet qui représente le mécanisme le plus répandu pour mettre en place des réseaux d'ordinateurs à grande échelle et homogènes. La méthode de base pour leur construction est majoritairement axée sur une architecture hiérarchique découplée où chaque sous-réseau est géré par un routeur connecté au reste de l'internet (cf fig. 1.2). Cette approche technologique permet d'offrir aux concepteurs un système prenant en charge la localisation des sites et le transport fiable et efficace des communications entre deux partenaires distants.

Avec l'arrivée des technologies sans fil, d'autres types d'architectures physiques sont apparues dont principalement les modèles à station de base et *ad hoc*. La station de base permet de retrouver l'architecture hiérarchique afin de proposer le plus simplement possible un accès transparent à un internet. Le mode *ad hoc* permet, quant à lui, de former dynamiquement des réseaux d'ordinateurs sans aucune architecture fixe et sans aucun gestionnaire de référence, *i.e* l'architecture est complètement décentralisée. La combinaison de ces deux techniques, à station de base pour donner accès à un internet, et *ad hoc*, pour faciliter son extension, va donner des architectures hybrides qui permettront d'offrir l'ubiquité aux unités mobiles. Dans de tels contextes, nous nous intéresserons aux éléments mobiles induisant une **mobilité matérielle**. Ils ne peuvent pas être gérés d'une manière hiérarchique classique et ne permettent plus d'obtenir un système garantissant, de façon permanente, la communication distante entre tout élément de l'environnement.

Dans la seconde partie du chapitre, nous avons présenté les différentes manières de prendre en considération les réseaux d'ordinateurs. Après avoir défini les grands schémas d'organisation pour la construction d'applications réparties, nous avons détaillé le modèle client/serveur qui est le plus largement répandu. Celui-ci se base sur des interactions distantes permettant à un client d'envoyer une requête de service à un serveur. Les différentes méthodes permettant de réaliser ces requêtes (envoi de message, appel à distance et invocation distante) ont montrées leur efficacité dans les environnements stables à grande échelle type Internet. Cependant, l'introduction de la mobilité physique, due aux technologies sans fil, apporte un degrée de dynamisme qui ne semble pas permettre à ces méthodes d'atteindre le même niveau d'efficacité mais elle peuvent être améliorées en introduisant la mobilité du code.

Dans ce cadre, les agents mobiles représentent un paradigme intéressant pour les environnements hybrides envisagés. En effet, leur autonomie leur permet de s'adapter facilement aux changements dynamiques de contexte et à l'hétérogénéité des éléments présents. De plus, ce paradigme permet de décrire facilement des comportements, utilisateurs ou applicatifs, difficiles à caractériser avec le modèle client/serveur classique, comme par exemple l'exploration, les déplacements d'unité mobile ou encore la représentation d'un utilisateur. Cependant, les agents mobiles se heurtent à de grandes difficultés de développement, se voient opposer leurs problèmes de sécurité toujours d'actualité et induisent un deuxième niveau de mobilité, *i.e* **la mobilité logicielle**.

Plusieurs articles de synthèse, construits sur l'avis de spécialistes du domaine de la mobilité, exposent la vision des futures directions que devraient suivre les agents mobiles [Mil99, KGR02, Gra04]. En résumé, ils viennent confirmer notre avis que les agents ne pourront pas être utilisés à grande échelle sans avoir résolu leurs problèmes de sécurité et qu'ils ne remplaceront pas intégralement le modèle classique du client/serveur qui s'applique parfaitement dans un grand nombre de cas. Ils viennent aussi confirmer que les agents mobiles, par leur capacité d'autonomie et d'adaptation, méritent d'être étudiés dans des environnements dynamiques décentralisés [MS99] constitués par les topologie hybrides.

Le chapitre suivant va permettre de faire une étude plus précise des agents mobiles en commençant par une étude comparative des principales plates-formes existantes. Nous mettrons en avant leurs intérêts majeurs en termes de conception et de développement. Puis nous regarderons plus précisément les méthodes d'interaction existant entre les agents. Enfin, nous terminerons par la présentation d'un ensemble d'applications envisageables pour les agents mobiles dans un tel contexte.

Les agents mobiles et leurs spécificités

2

Nous venons de présenter, dans le chapitre précèdent, le cadre général de notre étude comprenant deux niveaux de mobilité : un niveau physique, induit par la prise en charge des utilisateurs nomades dans les réseaux à architecture hybride, et un niveau logiciel introduit par l'utilisation des agents mobiles.

Dans ce chapitre, nous allons décrire plus en détails les moyens existants pour développer une programmation à base d'agents mobiles. Nous commençons par les intergiciels de développement, ou plates-formes à agents mobiles, en mettant l'accent sur les spécificités propres des agents mobiles apparaissant lors de la conception d'une application répartie. Puis nous continuons par la présentation des différents mécanismes d'interaction existants entre les agents et nous finissons par présenter quelques domaines d'applications existants.

2.1 Les plates-formes

Depuis l'apparition de la notion d'agent mobile, de nombreuses plates-formes ont été développées afin de faciliter la programmation d'applications structurées en termes d'agents mobiles. Elles devenaient si nombreuses, que la communauté des agents mobiles a créé un site permettant de les recenser [MAL], afin d'en faciliter la diffusion et dans le but inavoué d'en limiter la profusion. Plusieurs papiers de synthèse [FPV98, Ber00, Tho97] existent sur les principales plates-formes utilisées de nos jours. Ceux-ci nous serviront de base pour les descriptions à venir.

De notre point de vue, lorsque l'on parle de plates-formes, nous regroupons deux niveaux élémentaires. Le premier est le langage de programmation qui offre les primitives nécessaires à l'exécution, à la communication et à la mobilité, Il constitue le niveau de base. Le deuxième est l'intergiciel proprement dit qui offre des services de plus haut niveau, comme les annuaires, permettant aux agents la réalisation de leurs tâches.

2.1.1 Les langages sous-jacents

Un langage de programmation adapté aux agents mobiles doit offrir un ensemble de propriétés particulières décrites par [Kna96]. Il doit prendre en compte principalement :

La manipulation du code Le langage doit être capable d'identifier, d'envoyer, de recevoir et d'exécuter le code exécutable représentant un agent mobile. Le

langage proposera une abstraction de haut niveau pour réaliser une action de migration, comme une fonction *move on*.

L'hétérogénéité La construction d'un système distribué est caractérisé par l'hétérogénéité des éléments le composant. Le langage devra être indépendant de l'architecture matérielle et du système d'exploitation. L'agent doit pouvoir aller potentiellement sur tous les éléments du système. On voit ici que les langages interprétés semblent les mieux adaptés.

La sécurité En offrant un environnement à l'exécution de code étranger, on accroît la possibilité de recevoir des éléments indésirables au sein des systèmes. Le langage devra offrir un ensemble de mécanismes de protection, que ce soit par le cloisonnement ou une identification. Il s'agit du talon d'Achille des agents mobiles que nous avons détaillé dans la section 1.4.4.

L'exécution autonome Pour permettre une réelle mobilité, le langage ne doit pas exprimer des liens directs vers les ressources des sites visités. Ceci évite de gérer les problèmes de disparition des sites et facilite la représentation d'un utilisateur déconnecté.

La performance Dans un système multi-agents, chaque application utilise un ensemble d'agents s'exécutant en parallèle. Chaque agent nécessite un espace mémoire et des mécanismes de mise en route, de déplacement, etc, que le langage doit minimiser afin d'éviter les surcharges permettant ainsi un déroulement des applications dans un temps acceptable.

Le premier langage à proposer ces propriétés fut Téléscript [Dom96] de Général Magic qui fut suivi de nombreux autres tels Obliq [Car94], Safe-Tcl [Bor94], etc. Ils possèdent tous des qualités propres, comme la prise en compte de la mobilité forte par exemple, mais se voient supplantés par la popularité grandissante d'un langage à objets, à savoir Java. Une des premières plates-formes à utiliser cette technologie fut Mole [BHR97].

En effet, grâce à sa machine virtuelle (JVM) largement répandue, Java a su s'imposer pour la programmation d'applications réparties dans des environnements hétérogènes en permettant une indépendance envers les réseaux et les systèmes d'exploitation. La migration est supportée grâce au mécanisme de sérialisation pour le transport d'objets de sites en sites, la sérialisation étant la transcription de l'état et de la structure complexe d'un objet dans un format particulier.

Pourtant Java présente un inconvénient : il ne gère pas la mobilité forte. Malgré des travaux sur l'extension de la machine virtuelle [SBB+00, BHV01], c'est toujours au développeur de gérer les différents états rencontrés après une migration de ses agents.

Malgré cet inconvénient, nous retiendrons que Java est le principal langage utilisé pour le développement des plates-formes supportant les agents mobiles. Détailler plus avant le langage de programmation, en parlant par exemple de la gestion de la pile d'exécution, sortirait du cadre de notre étude. Nous invitons donc le lecteur désireux d'avoir plus d'information sur ce sujet à se reporter à la littérature sur le sujet.

2.1.2 Les normes

L'émergence des nombreuses plates-formes expérimentales a rendu nécessaire la proposition d'une harmonisation grâce à la standardisation des différents concepts communs pouvant

être identifiés. Cette normalisation devrait permettre à terme de rendre compatibles les différents systèmes.

On peut trouver à l'heure actuelle deux normes principales : il s'agit de la norme FIPA (Foundation for Intelligent Physical Agents) [FIP02] et de la norme MASIF (Mobile Agent System Interoperability Facility) [MBB+98].

Lorsque deux normes existent au sein d'une même technologie, elles ont une tendance naturelle à s'opposer alors que, dans notre cas, elles penchent plus vers la complémentarité due à la différence de leurs domaines d'origine.

MASIF La norme MASIF a été spécifiée par l'Object Management Group (OMG) qui se préoccupe généralement de l'hétérogénéité entre les systèmes, comme dans le cas de CORBA. Dans cette optique, le but, dans la norme MASIF, est de décrire les notions élémentaires permettant l'échange des agents entre différentes plates-formes. Pour ce faire, elle standardise la manière de gérer le code des agents[3], leur identification, la migration[4] et l'adressage local.

FIPA En revanche, la communauté d'origine de FIPA étant celle des systèmes multi-agents, plus proche de l'intelligence artificielle, elle va se situer à un niveau plus élevé *i.e* le niveau applicatif en décrivant les éléments nécessaires à la réalisation d'une application et principalement en détaillant la communication entre les agents. Le but est de décrire un ACL (Agents Communication Language), les ontologies et des protocoles de négociation permettant ainsi de définir parfaitement les interactions entre les agents.

Nous voyons donc que ces deux normes tendent plus vers la complémentarité que vers la divergence. C'est déjà le cas de FIPA qui a inscrit dans son planning l'intégration des règles de MASIF sur la gestion de la migration. En prenant un peu de recul, on peut extraire des deux normes les différents éléments devant apparaître dans l'élaboration d'une plate-forme [Rou02] et schématisés sur la figure 2.1 :

Les agents Ils sont définis par un identifiant unique au sein de la plate-forme et d'un état d'exécution comprenant le code et les données qu'ils transportent.

Les places C'est l'élément qui sera en charge d'exécuter le code de l'agent. Une place peut contenir plus d'un agent. Une place définit un environnement d'exécution vers lequel un agent sera orienté pour gérer son exécution.

Les agences Encore appelées « ports d'accueil » des agents, représentent les environnements où évoluent les agents. Une agence constitue le cœur de gestion de la plate-forme en assurant la délégation de l'exécution aux places qui la composent, l'administration, le contrôle, le transport effectif et la communication des agents grâce au système d'exploitation sous-jacent.

Les régions Il s'agit du regroupement de plusieurs agences, pas nécessairement de même type, appartenant à un même domaine d'expertise, d'activité ou autre. Le but est de faciliter les activités d'administration. On peut associer par exemple une région à des politiques d'accès aux services.

Les services On retrouvera un ensemble de services accessibles aux agents. Ils pourront être construits sur différents modèles, Client/Serveur ou agents, et

[3] création, suspension, reprise et terminaison
[4] par la sérialisation par exemple

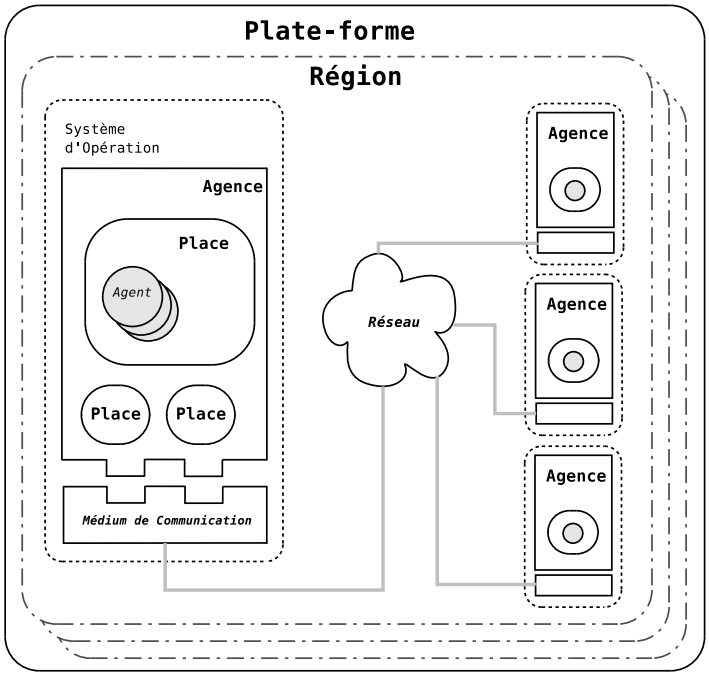


FIG. 2.1 – Les éléments d'une plate-forme d'agents mobiles

fourniront aux agents en visite les fonctionnalités recherchées. Citons pour exemple les services système (système de fichiers par ex.), de la plate-forme (annuaire par ex.), de sécurité, de négociation ou des services « métier » .

Après avoir exposé ces normes, nous allons maintenant présenter quelques plates-formes qui possèdent chacune une caractéristique précise qu'il faut prendre en considération pour notre modèle.

2.1.3 Étude comparative des plates-formes existantes

Si, d'un point de vue théorique, le langage de programmation est bien découplé de la plate-forme à travers la notion de place, la réalité ne respecte pas toujours ce critère et souvent ne nous permet pas de faire une vraie différenciation. C'est pour cette raison que nous allons nous intéresser, dans les présentations qui viennent, uniquement aux spécificités proposées au niveau des agents. Les plates-formes choisies sont présentées en fonction de la richesse des fonctionnalités de l'environnement qu'elle propose.

LIME

Le premier système que nous étudions est l'intergiciel basé sur Java nommé LIME (Linda In Mobile Environnement) qui propose une couche de coordination pour les agents en réutilisant le modèle Linda [MPR01, PMR99]. Ce n'est pas à proprement parler une plate-forme car elle ne prend pas en charge les différents éléments décrits par les normes. Elle se focalise plus sur la coordination qui peut être exploitée pour construire des applications réparties.

Comme l'indique son nom, LIME est basé sur le modèle de synchronisation Linda qui propose une communication à travers une mémoire partagée, appelée *espace de tuples*, qui est une collection de structures de données élémentaires appelées *tuples*. La synchronisation a lieu lors de la mise en jeu d'un tuple en accès ou en modification.

Ce type de communication est découplé (communication indirecte à distance) en *temps* et *espace*, *i.e* l'expéditeur et le destinataire n'ont pas besoin d'être connectés au même instant ni de connaître leurs emplacements respectifs. Cette caractéristique s'adapte particulièrement à un environnement mobile où les deux composantes d'une communication migrent dynamiquement.

Cette méthode de communication est utilisée dans LIME en associant un espace de tuples aux agents et en associant des mécanisme permettant la mise en commun des tuples. Chaque site possède une interface d'espace de Tuple (ITS) à laquelle pourront souscrire les agents en proposant les tuples à partager. Une ITS est l'union des tuples partagés par les agents locaux. Les agents pourront alors se synchroniser sur les tuples appartenant à l'ITS.

Pour étendre ce mécanisme aux communications distantes, les ITS peuvent être regroupées au sein d'une Fédération d'espaces de Tuples (FTS) créant ainsi une union d'ITS. Chaque FTS est gérée par un site leader, désigné par un mécanisme d'élection classique, dont le rôle est de prendre en compte les ITS entrants et sortants. Cette gestion centralisée permet de laisser la possibilité aux agents de dialoguer à distance et surtout permet de prévenir les agents de toute modification de l'environnement.

La partie qui nous intéresse dans LIME réside dans sa possibilité de définir une synchronisation avec un mécanisme très simple et surtout de permettre aux agents de contrôler les ressources mises en jeu en choisissant les éléments qu'ils souhaitent partager. Les sites sont réduits à un support d'exécution.

Mais avec les environnements très dynamiques que nous envisageons, la gestion centralisée par un leader des FTS et la communication distante ne sont pas adaptées. En effet, comment gérer la disparition du leader du FTS et, plus grave, comment garantir une synchronisation distante entre deux agents lorsque les liens de communication sont inconstant.

PLANGENT

PLANGENT est une plate-forme basée sur le langage Java s'intéressant à l'adaptation des agents durant leurs déplacements au sein des environnements dynamiques [ONI+97]. Il s'agit de l'intergiciel le plus proche de l'intelligence artificielle que nous avons étudié. Son but est de proposer aux agents des mécanismes pour modifier les objectifs intermédiaires nécessaires à la réalisation de leur tâche finale.

Un site supportant l'architecture PLANGENT inclut un intergiciel agent comportant le médium de communication, le gestionnaire d'information du site, le générateur d'agents et le gestionnaire de migration, et une partie système comprenant deux bases de données d'information, une pour l'environnement local et l'autre pour l'environnement distant. Les deux parties sont reliées et accessibles l'une vers l'autre.

Cette séparation permet de bien distinguer les éléments évoluant de ceux qui ne changeront pas durant l'exécution d'une application. En effet, l'intergiciel agent va rendre les services de base, sans se soucier de l'évolution de l'environnement, en répondant aux différentes requêtes énoncées par les agents, alors que c'est la partie système d'information qui prendra en compte les évolutions.

Les agents pendant la visite d'un site, pourront consulter les deux bases de données et ainsi connaître les éventuelles modifications de l'environnement. Grâce à un planificateur automatiquement incorporé, les agents pourront alors modifier leurs parcours en fonction des nouvelles informations récoltées.

Ce planificateur intégré aux agents détermine un parcours à suivre selon les buts à atteindre pour accomplir la tâche de l'agent. Ces buts correspondent principalement aux ressources nécessaires à la réalisation d'une tâche. Grâce à sa représentation de l'environnement, un parcours est calculé à l'initialisation de l'agent puis sera ensuite modifié au cours des différentes visites opérées par l'agent durant son exécution grâce aux nouvelles informations récupérées. Ce mécanisme d'adaptation modifie uniquement le parcours initial de l'agent mais pas l'agent lui-même.

Prenons un exemple simple d'une fin de journée où je dois récupérer les enfants, acheter le pain et rentrer à la maison. Avant de partir, j'établis le parcours le mieux adapté à mes besoins mais si, pendant mon parcours, j'apprends que certaines routes sont en travaux ou que la boulangerie est fermée, je modifie immédiatement mon itinéraire afin de prendre en compte les modifications.

Nous retenons ici que les agents sont capables de s'adapter durant leurs déplacements. Grâce aux bases de données sur l'environnement et au planificateur, les agents peuvent connaître les changements de l'environnement et s'adapter afin d'atteindre leurs buts. Cependant avec la mobilité matérielle, les liens de communication étant inconstants et les sites étant accessibles par intermittence, l'actualisation des bases de données, dont la méthode n'est pas précisée, semble difficile à mettre en place. Dans ce cadre, le planificateur, qui repose sur les bases de données des sites, met en route l'adaptation du parcours des agents avec des informations qui peuvent être erronées. Il serait donc intéressant de découpler le planificateur d'un composant système des sites afin de considérer uniquement les informations récoltées par les agents durant leurs déplacements.

TACOMA

Le projet TACOMA (Troms And COrnell Moving Agents) a été développé dans le but d'offrir différentes abstractions de haut niveau pour les agents. L'accent est mis sur un découplage, entre le langage et le niveau agent, semblable à celui introduit dans les normes à travers la notion de *place* [JvRS95, JLvR+02].

La première abstraction concerne la manière de capturer l'état d'un agent. Celui-ci est matérialisé par un *porte-documents* (PoD) associé à chaque agent, porte-documents qu'il transporte lors de ses migrations. Ce porte-documents est constitué d'un ensemble de classeurs contenant des suites de données engendrées par les actions passées de l'agent et nécessaires aux futures actions. Les opérations classiques (création, suppression etc.) sont disponibles, ainsi que des actions de sérialisation pour le transport qui peuvent être adaptées selon le langage sous-jacent.

La deuxième abstraction concerne la méthode de communication qui s'obtient grâce à une unique méthode *meet* sur un porte-documents. Lorsqu'un agent *A* souhaite dialoguer avec un agent *B* sur son porte-documents *PoD-A*, il effectue l'opération *meet B with PoD-A*.

Aucune autre abstraction n'est nécessaire pour mettre en place un premier système à base d'agents mobiles. En effet, chaque site contient un agent local *Rexec* exigeant deux classeurs dans le *porte-documents* de l'agent demandeur. Le premier est *HOST*, indiquant la destination désirée, et le second est *CONTACT*, contenant le nom de l'agent relatif à un interpréteur ou un

compilateur. La migration s'obtient alors par une simple action meet comme la suivante : *meet Rexec with PoD*. On peut exprimer de la même manière tout le système en termes d'agents et d'opérations *meet*.

La troisième abstraction porte sur la communication indirecte déléguée. En effet, pour enrichir le mécanisme de communication, chaque site contient au moins un *meuble de classeurs* permettant aux agents de laisser les classeurs qu'ils ne souhaitent plus transporter et qui pourront être utilisés par les futurs visiteurs d'un site. Cette méthode permet de définir une communication indirecte et asynchrone entre des agents qui visiteront le même site à des instants différents.

Nous retenons de ce projet les deux abstractions qui permettent d'exprimer l'intégralité du système en terme d'agents et la communication indirecte déléguée. Dans un environnement où l'accessibilité accordée aux sites est limitée à cause de leurs déplacements fréquents, il est important de pouvoir mettre les agents au cœur du système et de leur permettre de communiquer grâce à un intermédiaire.

Néanmoins, nous voyons que la communication indirecte reste basée sur les sites qui possèdent, répétons le, une accessibilité intermittente et par conséquent limitée. Pourquoi ne pas incorporer cette méthode de communication directement dans des agents ? Ils serviront, alors, d'intermédiaire en nous permettant d'avoir une totale séparation entre le système et les sites supports[5].

Un problème crucial venant de ce type de communication différée, qui apparaîtra lors de la construction des applications, est la vérification de la validité des informations récupérées grâce aux intermédiaires. Si elle ne fait pas l'objet de l'étude sur TACOMA, nous aborderons cette problématique, d'une manière applicative, dans la partie expérimentation (cf chapitre 6) lors de la mise en place de mécanismes de rumeurs qui appliquent les méthodes de communication des fourmis dans le cadre de l'informatique répartie [D+87].

JADE

JADE (Java Agent DEvelopment Framework) est un intergiciel construit sur Java afin de permettre une programmation multi-agents simplifiée en prenant en compte la norme FIPA [BPR99]. C'est un intergiciel très abouti qui offre un environnement graphique pour l'aide au développement et à l'administration.

La préoccupation de JADE étant de respecter la norme, on retrouve les différents éléments architecturaux décrits dans la présentation des normes résumée sur la figure 2.1. Ne supportant qu'un seul langage de programmation, un élément appelé *container* est construit comme une agence contenant une unique place exécutant une machine virtuelle Java. Une région est alors définie par un ensemble de containers distribués correspondant concrètement à la plate-forme JADE.

Cette architecture est distribuée et hiérarchique au travers d'un *Main-container* servant de container de départ auquel viendront se rattacher l'ensemble des autres containers répartis. Il existe aussi une possibilité d'associer plusieurs plates-formes en reliant leur *Main-container* mais en limitant leurs fonctionnalités. En effet, les régions définies par les *Main-container* autorisent uniquement l'échange de messages mais pas la migration des agents. Tous ces éléments sont gérés par des agents qui représentent l'entité de base de la plate-forme.

[5] C'est ce que nous verrons plus loin dans la section 2.2.4

Les agents respectent la notion d'une exécution par le déroulement d'une succession de comportements que nous avons décrite dans la partie 1.3.2. C'est l'ensemble des comportements et de leurs transitions qui définissent la fonction globale d'un agent.

Les agents peuvent migrer au sein d'un même ensemble de containers. Cette migration, qui peut être proactive ou réactive, est réalisée par le *main-container* en se limitant aux containers sous sa responsabilité. Malgré des travaux en cours [ARB03] sur l'utilisation des messages ACL pour déplacer les agents, la migration inter plates-formes, *i.e* inter *main-container*, n'est pas réalisable à l'heure actuelle.

Les agents peuvent uniquement dialoguer par l'envoi de messages asynchrones respectant la norme ACL. Ces messages incorporent de la sémantique, définie par une ontologie, sur les données transportées permettant ainsi de garantir une compréhension entre les agents lors des phases de dialogue.

JADE propose aussi la possibilité d'intégrer des services décrits et référencés dans un annuaire de pages jaunes. Cette description est réalisée grâce aux ontologies que l'on trouve dans les messages et s'accompagne d'une description du protocole que doivent suivre les clients du service demandé. C'est une notion proche des « WebServices ». Pour aider les concepteurs d'agents et de services, un outil de génération automatique d'ontologies est fourni.

Grâce à une conception des agents basée sur la notion de comportement, à une expressivité de la communication basée sur les ontologies et à ses outils d'aide au développement, JADE offre un environnement complet pour la réalisation d'applications réparties construites sur un ensemble d'agents mobiles.

Mais son architecture centralisée et la limite apportée à la migration des agents représentent une limite à l'adaptation de JADE aux environnements dynamiques. Comme pour LIME, on ne peut se satisfaire d'un gestionnaire centralisé fragilisant l'intégralité du système s'il venait à disparaître. De plus, pour garantir leur autonomie vis-à-vis du système, les agents doivent pouvoir se déplacer suivant leurs besoins et même dans des plates-formes différentes.

2.1.4 Conclusion de l'étude

Nous avons décidé de présenter l'étude détaillée de ces différentes plates-formes car, selon nous, chacune permettait de mettre en avant des caractéristiques que nous souhaiterions trouver dans un intergiciel pouvant s'adapter dans un environnement dynamique.

Nous avons regroupé les principales caractéristiques de ces plates-formes dans le tableau récapitulatif 2.1 afin d'en avoir un aperçu rapide. Pour ce tableau, nous avons regroupé, à titre de comparaison, les différentes plates-formes que nous avons étudiées en donnant uniquement quelques informations synthétiques.

Les domaines d'application de ces plates-formes reposent généralement sur des environnements distribués dont l'architecture possède un noyau formé d'un ensemble de sites fixes. Les éléments nomades sont pris en compte en les rattachant à des point d'accès au noyau. Ce type d'architecture permet parfaitement de concevoir des intergiciels ayant les sites comme éléments de base et permettant la mise en place d'une administration plus ou moins centralisée.

Mais nous avons vu dans la partie 1.1.3, que les topologies hybrides permettront aux architectures de s'étendre sans avoir de connexions directes au noyau. Cette topologie basée principalement sur l'emploi de la technologie sans fil impose une accessibilité intermittente des sites et ne permet plus de les prendre comme référentiel. Cette constatation remet en cause la conception classique des intergiciels.

Plate-forme	Agents		Communication		Langage	Réf
	Réac/Pro	Migration	sync/async	locale/distante		
ARA 1.0a	Proactif	Forte	sync	locale	Tcl	[PS97]
Aglets	Proactif	Faible	les deux	les deux	Java	[Jon02]
D'Agents	Proactif	Forte	les deux	les deux	multiple	[GCK+02]
JADE	les deux	Faible	async	les deux	Java	[BPR99]
JavAct	Proactif	Faible	les deux	les deux	Java	[ALP04]
LIME	Supportées[6]		sync	les deux	Java	[PMR99]
PLANGENT	Proactif	Faible	sync	locale	Java	[ONI+97]
TACOMA	Proactif	Faible	les deux	locale	multiple	[JLvR+02]
Telescript	Proactif	Forte	sync	les deux	Telescript	[Dom96]

TAB. 2.1 – Récapitulatif des plates-formes existantes

Les éléments de base de l'intergiciel idéal

Pour permettre à un intergiciel de s'adapter aux topologies hybrides, nous devons reconsidérer les méthodes de conception d'une plate-forme tout en gardant certains éléments propres aux systèmes multi-agents que nous détaillons maintenant :

La base de la plate-forme est l'agent Les sites ayant une accessibilité limitée, un intergiciel doit s'appuyer exclusivement sur les agents qui pourront s'adapter aux changements de l'environnement. Les sites proposeront uniquement un support d'exécution. Cette conception n'interdit pas des mécanismes de distribution classique des sites avec un noyau fixe, mais les agents les considèrent toujours comme des éléments pouvant disparaître.

Une communication locale Les déplacements fréquents des sites ne permettent pas de mettre en place une communication distante qu'elle soit synchrone, car pouvant être interrompue trop fréquemment et/ou totalement, ou qu'elle soit asynchrone, car la double mobilité site/agent ne permet pas de garantir la livraison d'un message dans un délai raisonnable. Par conséquent, la communication s'effectue uniquement entre des agents locaux afin de garantir un déroulement sans interruption. Cependant une exception confirme la règle en permettant aux agents gérant la migration d'établir des liaisons distantes le temps du déplacement des agents.

Une communication déléguée Pour permettre une communication plus riche, les agents ont la possibilité de dialoguer grâce à des agents intermédiaires. Ces agents pourront être mobiles ou statiques selon les besoins. Ce dialogue se fait toujours entre éléments locaux. On peut aisément imaginer la conception d'un agent réalisant un bureau de poste qui sera local ou mobile.

[6] LIME propose un médium de synchronisation. Il supporte donc les différents types d'agents en fonction du système sous-jacent

Une communication expressive Les communications sont basées sur un ensemble d'ontologies permettant de définir précisément la sémantique des échanges. Comme nous l'avons vu dans la norme FIPA, l'utilisation des ontologies permet de garantir que deux agents se comprennent correctement lors des phases de dialogue.

Une migration assumée L'agent étant le cœur du système, l'intergiciel doit permettre aux développeurs de choisir le type de migration nécessaire à la réalisation de leurs applications. Les déplacements sont effectués grâce à un service de migration auquel s'adressent les agents, lorsqu'ils souhaitent changer de site, ou le système, lorsqu'il doit faire bouger un agent.

Même dans le cas où l'agent gère de manière autonome ses déplacements, la migration ne donne pas des droits de déplacement sans limite. Lorsque le service de migration reçoit une requête de migration d'un agent, il statue selon des règles établies s'il accepte ou non la demande puis envoie le verdict à l'agent. Cette méthode permet de garder un contrôle d'accès à certains sites critiques mais ne remet pas en cause la gestion des déplacements propres de l'agent, il s'adaptera selon les verdicts du service de migration.

En fait l'intergiciel propose un ensemble de types de migration possibles, portant sur la prise de décision de la migration et sur sa perception, qui seront utilisés par le développeur lors de la conception des agents. Le type n'est pas figé durant le cycle de vie d'un agent, il pourra s'adapter selon les besoins. Prenons l'exemple d'un agent autonome qui arrive sur une grille de calculs composée d'un ensemble de sites strictement identiques, l'agent pourra décider alors de laisser le système prendre le contrôle de ses migrations le temps qu'il termine ses calculs.

Un annuaire de services L'intergiciel doit aussi proposer un annuaire de pages jaunes permettant de regrouper les services atteignables localement. Il propose les fonctions habituelles d'enregistrement, de modification, de suppression et de recherche. En fait, ces services sont représentés localement par un agent commis qui sera contacté par le client qui effectue une demande.

Il est nécessaire de définir des protocoles d'interaction entre les agents qui seront présents dans les descriptions inscrites dans l'annuaire. Un client qui souhaite utiliser un service reçoit le protocole détaillé à employer pour l'utilisation du service.

Les services étant réalisés eux aussi par des agents, ils peuvent se déplacer, déléguer ou travailler localement en fonction des besoins nécessaires à l'accomplissement du service. La manière de récupérer les résultats est précisée dans le protocole d'utilisation d'un service.

L'environnement de développement idéal

Maintenant que nous avons décrits les éléments de base nécessaires que doit proposer un intergiciel afin de s'adapter à des topologies hybrides, nous présentons un ensemble d'abstractions et d'outils devant être intégrés par l'environnement de développement proposé aux concepteurs d'applications réparties.

Un modèle comportements/transitions L'intergiciel doit permettre au développeur de maîtriser les différents états atteints par ses agents au cours de leur exécution. Les changements d'états interviennent principalement lors de la terminaison d'une tâche ou juste avant une migration et doivent être clairement définis lors de la conception. Ceci se réalise parfaitement

en proposant un intergiciel qui respecte le modèle des agents définis grâce à un cycle de vie référent et un ensemble de comportements/transitions (cf. 1.3.2).

Une migration simple La migration étant une méthode cruciale pour l'adaptabilité, un ensemble de mécanismes simples sont proposés au développeur pour sa mise en œuvre. Le développeur possède un ensemble de fonctions permettant de préparer un déplacement, en se séparant de données inutiles par exemple, de migrer en donnant juste la destination et de mettre en place son état post migratoire.

Nous choisissons une migration par expression explicite de la destination pour que le développeur garde le contrôle de ses déplacements. Le développeur peut implémenter une migration au hasard en choisissant dans un ensemble de sites atteignables qu'il peut obtenir depuis le service d'environnement, mais il s'agit d'un choix délibéré.

Des mécanismes de synchronisation Les applications étant construites sur un ensemble d'agents, le développeur dispose de mécanismes de synchronisation permettant de contrôler les phases d'échanges. Ces synchronisations peuvent tout aussi bien s'opérer sur un espace de données partagées, un tableau blanc par exemple, que sur la délégation d'une tâche. Ceci permet de définir certaines contraintes lors des phases de coopération, comme éviter qu'un agent ne migre par exemple. .

Une connaissance de l'environnement Un service, référencé auprès de l'annuaire, permet de renseigner le développeur de l'état actuel de l'environnement. Ce service donne la possibilité au développeur, s'il le souhaite, de connaître les changements dans l'environnement où évoluent ses agents et ainsi de permettre la mise en place de comportements spécifiques.

Des outils graphiques Un ensemble d'outils d'aide aux développement, comme ceux de JADE, permet au développeur de suivre facilement le déroulement de leur applications.

Au final

Après avoir étudié les deux normes pour les systèmes multi-agents et les plates-formes existantes, nous avons constaté que l'adaptation aux futures architectures hybrides n'est pas réalisable dans l'état actuel des intergiciels. Nous avons identifié les raisons de cette incompatibilité à partir desquelles nous avons présenté les différents éléments nécessaires à un modèle d'intergiciel capable de s'adapter aux environnements dynamiques en question.

La plate-forme se rapprochant le plus de notre modèle est sans conteste JADE qui possède quasiment tous les éléments désirés, mais reste limitée sur un point crucial : la migration des agents est gérée de manière centralisé et limitée à un même ensemble de sites. La communauté grandissante de ses utilisateurs et surtout l'activité du projet, qui est continue et surtout soutenue par de nombreux partenaires industriels, nous laissent envisager favorablement que les manques seront rapidement comblés.

Maintenant que nous connaissons les possibilités offertes par un l'intergiciel idéal, nous allons nous intéresser au niveau applicatif afin de regarder les particularités du développement d'applications basées sur les agents mobiles.

2.2 La coopération

En nous basant sur des intergiciels ne proposant que des mécanismes de communication locale entre agents s'exécutant sur des sites en déplacement, on impose au développeur d'envisager une méthode de conception autre que le client/serveur classique fondée sur des communications distantes fiables.

Le concepteur doit maintenant définir la tâche globale d'une application comme un ensemble de sous tâches réalisées par des agents mobiles. Durant leur exécution, les agents de cet ensemble doivent communiquer afin de connaître l'état d'avancement de la tâche générale, pour en connaître la terminaison par exemple, ou pour échanger des données sur les actions accomplies, comme le résultat d'un calcul. Ce sont ces phases de dialogue que nous nommons coopération.

La raison d'être principale de la coopération est d'aider le concepteur à construire une application plus performante dans les environnements dynamiques. Dans cette partie, nous allons présenter les différentes approches que nous avons rencontrées et nous les mettons en corrélation avec l'intergiciel que nous avons présenté. Mais avant de commencer nous devons cadrer précisément le concept de coopération qui est utilisé dans plusieurs domaines.

2.2.1 Les différentes vues d'un système multi-agents

Nous avons vu dans la partie 1.3.1 que le terme *agent* est commun à plusieurs communautés distinctes. Il en va de même pour les définitions du *système multi-agents* et, par suite, du terme *coopération* que nous devons donc préciser. Pour ce faire, nous nous appuyons sur le livre [Fer95] et sur l'état de l'art de la thèse [Gau04].

Dans les domaines proches de l'intelligence artificielle, l'agent est défini comme une entité physique ou virtuelle capable d'agir sur son environnement de manière autonome en fonction de ses perceptions et de ses connaissances partielles. L'agent pouvant être une entité physique, comme un avion ou un robot, il évolue dans un environnement réel possédant des objets localisés géographiquement grâce à une localisation précise. Il peut alors modifier physiquement l'environnement en déplaçant les objets.

Dans ce cadre, un système multi-agents impose la prise en compte de problématiques organisationnelles et collectives portant sur les entités, la dynamique et la structure. On peut alors définir un système multi-agents comme un ensemble d'objets et d'agents autonomes ayant pour but de fournir une fonction collective. Les mécanismes permettant d'obtenir cette fonction prennent en compte les enjeux sociaux entre agents comme la coopération.

Prenons l'exemple d'une masse de 300 kg devant être déplacée par plusieurs robots possédant chacun une capacité limitée à 100kg. C'est le regroupement de plusieurs agents qui permettra de satisfaire le but global et leurs objectifs propres. C'est de cette manière qu'on définit une coopération dans le domaine des systèmes multi-agents. De plus, on peut constater sur cet exemple que le rapprochement de plusieurs agents n'est pas suffisant en soit, car dans le cas d'un cylindre, mettre tous les robots à une seule extrémité peut s'avérer inefficace. Il faudra alors organiser les agents dans l'environnement afin qu'ils puissent accomplir leurs tâches.

Pour les systèmes multi-agents définis ici, les notions de coopération et d'organisation sont étroitement liées car une coopération désorganisée peut rendre irréalisable le but à atteindre et la mise en place d'une bonne organisation nécessite une coopération initiale. Dans ce cas la

coopération se définit comme un regroupement organisé d'agents qui doivent agir ensemble pour accomplir leurs objectifs.

Même si la frontière entre les systèmes multi-agents présentés ici et les systèmes distribués se réduit [CLMZ01], nous nous démarquons assez franchement des concepts que nous venons d'énoncer. Pour commencer, dans notre cas les agents sont uniquement «logiciels» et sont les uniques éléments référencés, il n'y a pas d'objet physique réel.

Ainsi, nous nous intéressons seulement aux programmes logiciels travaillant sur un modèle virtuel d'un environnement physique, ce modèle donnant la capacité aux agents de s'exécuter et nous le désignons par le simple terme ***environnement***. Les agents, représentés par ces programmes, n'ont pas le but explicite de modifier physiquement l'environnement et cherchent simplement à utiliser les ressources informatiques mises à disposition, allant du processeur aux services applicatifs, afin d'accomplir la tâche pour laquelle ils ont été créés.

De plus, en utilisant des agents pour représenter les ressources, nous obtenons des environnements formés uniquement d'un ensemble d'agents en cours d'évolution, ne subissant aucune modification physique volontairement engendrés par les agents. Dans ce cadre, on peut définir la localisation d'un agent comme le site en cours de visite, *i.e* en train de l'exécuter, et on peut s'affranchir de la notion d'organisation qui constituerait à placer les agents sur des sites spécifiques alors que nous les définissons uniquement comme des supports d'exécution quelconques, c'est à dire que sous ces hypothèses, seul l'indentifiant de site les différencis.

Avec les environnements et applications formés uniquement d'agents logiciels, nous avons besoin d'une plate-forme permettant de gérer l'exécution, le nommage, le médium de communication etc. Ces différents éléments de gestion nécessaires à l'évolution des agents sont regroupés au sein de ce que nous nommons l'intergiciel.

Nous définissons alors un système multi-agents comme la composition d'un intergiciel, offrant le contexte d'exécution, et des agents en cours d'évolution. Ces systèmes multi-agents peuvent regrouper en leur sein plusieurs applications distinctes qui n'ont aucun rapport les unes avec les autres. Avec cette définition, nous appelons ***coopération*** les phases de dialogue, entre paires d'agents, correspondant aux demandes de services, à leur réalisation ou aux échanges de données.

En utilisant une localisation des agents basée sur le site en cours d'exécution et une coopération comme nous venons de la préciser, la notion d'organisation, décrite dans le cadre de l'intelligence artificielle, s'obtiendrait en choisissant de manière explicite la répartition des agents en cours de coopération sur l'ensemble des sites de l'environnement afin qu'ils puissent accomplir au mieux leurs tâches.

Cependant, nous constatons que cette répartition dépend à la fois d'un important nombre de paramètres propres au système, par exemple la répartitions de charge, la minimisation des déplacements ou la limitation d'utilisation de la bande passante, et à la fois de paramètres applicatifs propres aux nombreux types d'applications réalisables. En ayant conscience que le choix de la répartition a un effet non négligeable sur la performance des applications, ces deux facteurs nous amènent à conclure que l'organisation ne fait pas partie du cadre de notre étude. Nous décidons alors de ne pas l'aborder dans le reste de ce document afin de nous concentrer sur les effets de la coopération.

Maintenant que nous avons bien cadré les notions de système multi-agents et de coopération, nous allons présenter les différents mécanismes que nous avons rencontrés pour réaliser les phases de coopération permettant d'améliorer les performances des applications réparties dans les environnements dynamiques.

2.2.2 La coopération entre sites

La coopération entre sites se retrouve principalement dans le modèle Client/Serveur classique que nous avons exposé dans la partie 1.2.2. Dans ce cas, un site client adresse une demande de réalisation de service grâce à l'établissement d'une communication distante avec un site serveur détenteur du savoir faire. Cette communication se réalise grâce à un protocole proprement défini prenant en compte la méthode de dialogue, allant de l'envoi de message jusqu'à l'appel de méthode à distance.

On peut citer à titre d'exemple les intergiciels respectant la norme CORBA qui permettent de distribuer des d'objets sur différents sites interconnectés afin de réaliser des services que l'on appelle par le mécanisme d'invocation de méthode à distance.

Cette méthode est depuis longtemps la mieux maîtrisée et reste parfaitement adaptée lors de la mise en place d'applications reposant sur une architecture réseaux stable où l'accessibilité des sites n'est pas remise en cause. Son principal atout est de permettre de s'affranchir des problèmes d'hétérogénéité.

Par contre, elle ne s'adapte pas lors du passage aux environnements dynamiques où les sites se déplacent si fréquemment qu'ils offrent une accessibilité limitée et ne permettent pas d'établir des communications distantes garanties sans interruption.

2.2.3 La coopération entre sites et agents

En donnant l'autonomie aux agents, on définit un découplage entre l'application et les sites d'exécution qui permet de répartir naturellement les tâches en affectant, généralement, aux agents la gestion de leurs applications créatrices et aux sites la gestion des ressources utilisées par les agents. C'est pour cette raison que l'on retrouve principalement des coopérations, entre les agents et les sites, portant sur l'optimisation de l'utilisation et la localisation des ressources.

Pour ce qui est de l'utilisation des ressources, un site peut donner à un agent l'ensemble de propriétés propres d'une ressource, comme la puissance du processeur, mais il l'accompagne aussi des contraintes d'utilisation, comme le temps de calcul qui lui sera accordé. Selon les données récupérées par l'agent, il évalue, selon ses propres critères, l'intérêt d'utiliser la ressource considérée et adapte son comportement en fonction.

Un exemple de ce type de coopération est la mise en place des notions, empruntées au domaine économique, d'offres et de demandes [BKR98]. Ces notions permettent aux sites d'adapter les propriétés des ressources offertes aux agents : par exemple plus une ressource sera demandée, plus les contraintes d'utilisation seront élevées. Mais ce type de coopération est à sens unique car les agents ne participent pas volontairement à l'adaptation du site face aux demandes. Ce sont les sites qui collectent les demandes et effectuent les modifications nécessaires.

Dans un environnement distribué, les ressources sont réparties sur l'ensemble des sites et chacun possède une représentation partielle et locale de cette répartition. Dans un environnements dynamique, cette répartition n'est pas figée dans le temps et est modifié selon les besoins. Dans ce cas, lorsqu'un agent cherche une ressource, il demande localement au site où elle se situe et va se déplacer pour aller l'utiliser. Si cette ressource a été déplacée, comme un fichier par exemple, et que le site de l'agent n'a pas été informé avant la demande, l'agent effectue alors le déplacement vers un site qui ne possède pas la ressource mais qui détient une infor-

mation plus récente de sa localisation. Il répète alors sa demande jusqu'à finalement trouver la ressource recherchée. Il s'agit là du mécanisme de suivi de lien de poursuite.

L'idée est de mettre plus activement à contribution les agents car lorsqu'il trouve la ressource désirée, il possède alors l'information la plus récente concernant sa localisation et peut alors informer son site créateur qui mettra à jour sa représentation de l'environnement. Ce mécanisme étudié par [JHK99], permet d'améliorer la connaissance locale de l'environnement en donnant aux sites une représentation plus proche de la réalité et ceci grâce à un surcoût limité car les agents explorent de fait l'environnement. Les sites n'ont pas à le faire eux-mêmes, par des mécanismes coûteux d'inondation par exemple. En mettant à jour le site initial ou final de l'agent [SM98], on met en place une coopération site/agent permettant d'améliorer la connaissance de l'environnement et par conséquent la pertinence des déplacements des agents.

Nous voyons que ce mécanisme de lien de poursuite repose sur l'idée qu'un agent arrivera toujours à connaître les différents déplacements d'une ressource en suivant ce lien de poursuite géré par les sites. Mais avec les environnements que nous envisageons, où les sites ne garantissent pas leur accessibilité à cause de leurs fréquents déplacements, l'intégrité du lien de poursuite n'est plus assuré et peut rendre irréalisable la recherche d'une ressource.

Nous en concluons que ce type de coopération s'adapte parfaitement aux applications multi-agents reposant sur une architecture physique relativement stable et permettant le déplacement de ressources (pour la répartition de charge par exemple). Dans le cadre des environnements dynamiques, elle reste aussi valable pour la gestion locale des ressources d'un site, mais pas pour la gestion de la représentation de l'environnement qui ne devra pas dépendre de sites dont l'accessibilité est intermittente.

2.2.4 La coopération entre agents

Pour la coopération entre agents, nous nous intéressons uniquement aux interactions locales pour la simple raison que dans le cas contraire nous reviendrons à la coopération entre sites que nous avons déjà décrite. Nous abordons pour commencer les différents choix de migrations et le concept de coordination, nécessaire aux interactions locales, avant de présenter plus précisément les différents mécanismes de coopération entre agents.

Migration et coordination

En choisissant une coopération locale et des agents mobiles, nous devons prendre en considération le problème de localisation d'une rencontre. En effet, les agents doivent commencer par se trouver sur un site commun permettant d'engager leurs phases de coopération. Le choix du site support va dépendre du type de rencontre désirée par les agents, celle-ci pourra être volontaire ou effectué au grè des déplacements. Ces deux types de rencontres envisagées vont induire une politique de déplacement propre à chaque agent qui sera différenciée en deux grandes classes de migrations.

La première porte sur le cas d'une rencontre volontaire où les agents souhaitent coopérer avec un partenaire parfaitement identifié mais dont la localisation est incertaine. Sous ces hypothèses, les agents s'informent de l'état de l'environnement et choisissent la prochaine destination correspondant au site permettant de se rapprocher le plus possible du partenaire cible. Dans le cadre d'une rencontre volontaire, la politique de déplacement est appelée *migration ciblée*.

La deuxième classe englobe les rencontres involontaires où les agents se retrouvent sur un site quelconque et souhaitent coopérer avec les agents présents. Dans ce cas, les agents se déplacent au hasard vis à vis des autres agents, s'informent des partenaires éventuels une fois arrivés sur un site et engagent les coopérations possibles. Dans le cadre d'une rencontre involontaire, la politique de déplacement est appelée *migration libre*.

Ces deux types de migration nous permettent de classifier les déplacements mais ne limitent pas la possibilité de définir des politiques de migration intermédiaires. Le choix de la politique est déterminé par le développeur en fonction des besoins de son application. Il doit les identifier avec la plus grande précision possible afin d'éviter les problèmes de localisation, comme des agents qui se cherchent mutuellement en se courant après sans arriver à se rattraper.

Intuitivement nous pouvons énoncer que la *migration libre* s'applique principalement aux services systèmes s'occupant de la gestion du contexte où les éléments changent régulièrement, au contraire des applications dites métier qui vont nécessiter d'utiliser des services parfaitement identifiés et vont plutôt utiliser la *migration ciblée*. Différentes études ont été menées dans la cadre de la migration au hasard afin de l'appliquer dans la gestion de réseau [Spi76, DS84, MR95], mais la migration ciblée n'a pas encore été examinée en profondeur dans les environnements dynamiques.

Nous voyons bien ici que la réalisation des rencontres n'est absolument pas triviale et qu'elle va nécessiter un important investissement de conception de la part du développeur en fonction de l'environnement dans lequel évolueront ses applications [MX03]. Ceci est surtout dû au manque d'expériences existantes sur le rapport entre les migrations des agents mobiles et la fréquence de déplacements des sites. Ce problème constitue une grande partie de cette thèse que nous abordons dans le chapitre 3.

Ceci nous conduit au second problème inhérent à la coopération locale entre agents mobiles : garantir que les interactions arrivent jusqu'à leur terme avec le minimum d'interruptions possibles. En effet, avec cette difficulté d'établir des rencontres, surtout pour celles initiées volontairement, nous devons éviter au maximum de les remettre en cause en raison d'une interruption prématurée de la coopération.

Si nous éliminons les perturbations propres à tout système informatique (erreurs du système d'exploitation, panne de courant ...), nous devons identifier de manière précise ce qui peut être à l'origine d'une interruption prématurée de la coopération. Or, nous avons défini la coopération comme une phase de dialogue entre pairs et, en fait, seuls deux événements sont capables de mettre fin à un échange en cours : soit le lien de communication se rompt, soit un participant se retire sans prévenir.

En choisissant une coopération locale, le lien de communication est remis en cause uniquement en raison d'une erreur de l'intergiciel, se produisant assez rarement pour que nous puissions écarter ce cas et nous focaliser sur le départ prématuré d'un pair. En effet, les participants étant des agents mobiles, un départ inopportun est plus que probable et nous devons proposer un mécanisme permettant de restreindre son apparition.

Pour résoudre ce problème, nous introduisons la notion de *coordination* entre agents que nous définissons comme la description des différentes étapes respectées par les intervenants lors d'une coopération afin d'en garantir le déroulement intégral et ininterrompu. Nous pouvons l'apparenter à une sorte de protocole d'utilisation. Cette coordination offre un découplage entre le comportement général des agents et celui qu'ils adoptent durant les phases de coopé-

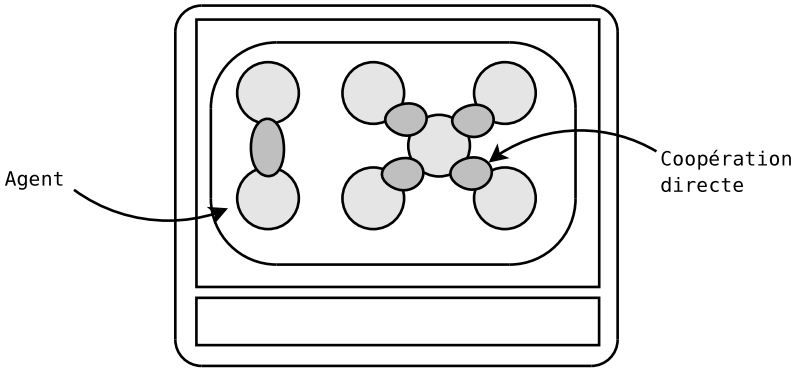


FIG. 2.2 – Coopération directe

ration [MPR00], ceci nous permettant d'éviter le déplacement dû à sa tâche globale alors qu'il est encore en cours de coopération.

Le concepteur devra donc, en plus du comportement global, définir pour chaque coopération la coordination nécessaire à son utilisation, mais il devra aussi définir une politique générale de coopération applicable aux agents. Elle correspond à la gestion locale de l'ensemble des coopérations proposées permettant à l'agent de garder le contrôle de son évolution par rapport à sa tâche globale et de ne pas stagner dans une suite sans fin de coopérations, par exemple on pourra limiter les nombres de coopérations séquentielles et/ou parallèles, désigner l'identification de ses partenaires, etc.

Maintenant que nous avons présenté ces concepts de politique de migration, de coordination et de politique de coopération, nous possédons toutes les notions permettant d'établir une coopération, de garantir son déroulement ininterrompu et d'assurer l'évolution de la tâche principale de l'agent. Avec ces trois éléments introduits, nous pouvons étudier les méthodes existantes qui permettent de réaliser la coopération entre les agents mobiles.

Coopération directe

En autorisant uniquement les communications locales, la coopération entre agents se rapporte à deux entités d'un même système souhaitant communiquer. L'intergiciel peut alors réaliser concrètement la communication grâce à l'ensemble des moyens de communication offerts par l'environnement d'exécution (pipe, signaux, appel de méthode, ect). En plus de cette communication, l'intergiciel doit offrir des mécanismes de haut niveau permettant de mettre en place la coordination et la politique de coopération.

Cependant en réalisant localement la coopération, on se place, de fait, dans un contexte plus fiable, c'est à dire sans les problèmes de communication liés à l'utilisation du réseau. Cette fiabilité permet de baser la coopération, au niveau des agents, sur des appels de méthode. Ainsi, nous pouvons définir la coopération directe entre deux agents comme une séquence d'appels de méthode supervisée par un mécanisme de coordination.

Ce choix de communication par pair ne limite pas les différents modes de coopération envisageables (cf figure 2.2). On peut très bien imaginer de mettre en place une coopération basée sur l'idée d'un tableau blanc anonyme qui sera réalisé par un agent tiers auquel s'abonnent les participants, chaque agent engageant une coopération deux à deux avec le gestionnaire du tableau. Ce type de mode de coopération permet de ne pas se limiter à des échanges bipolaires

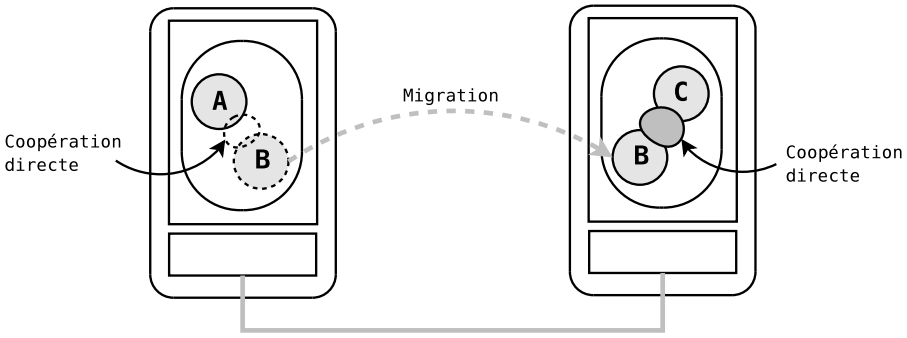


FIG. 2.3 – Coopération indirecte

et peut étendre les phases de dialogue à tout un groupe d'agents. En poursuivant l'idée du tableau blanc, il peut tout à fait servir de centre pour le dépôt de services où chacun ajoute ses demandes et récupère des requêtes à satisfaire.

Nous voyons qu'il est tout à fait possible d'imaginer un vaste panel de coopérations au niveau applicatif, mais qu'il est nécessaire que le système aide les agents à se faire connaître de leurs pairs et à publier les coopérations proposées avec les coordinations associées. Pour ce faire l'intergiciel proposera un service d'annuaire local, accessible d'une manière identique sur chaque site de l'environnement grâce à un agent local parfaitement identifié. Ce service nous donne un point de départ pour chaque agent, un peu à l'instar du service de nommage de CORBA, et permet aux agents de réaliser toutes les actions de publication et de recherches désirées.

Mais cette coopération directe nous permet-elle d'améliorer le comportement général de nos applications ? Prenons le cas de la découverte de la topologie d'un réseau [MKM99], la tâche principale des agents est de parcourir les sites qu'il connaît afin de récupérer d'autre éléments du réseau qu'il ne connaît pas, dans le but de compléter sa cartographie. Lorsque les agents agissent seuls, ils doivent tous parcourir l'ensemble du réseaux, alors que s'il coopèrent, ils se répartissent les sous-ensembles du réseau à explorer et ainsi n'en parcourent pas l'intégralité. En répétant cet algorithme à chaque rencontre, les agents convergent beaucoup plus rapidement vers la découverte totale de la topologie qu'un ensemble d'agents isolés.

Lorsque les agents utilisent cette coopération directe, ils tirent bénéfice du travail déjà réaliser par leurs homologues et peuvent alors affiner leurs tâches futures, permettant ainsi d'améliorer le comportement général de l'application qu'ils représentent. Ce qui est intéressant de noter, dans le cas de la cartographie d'un réseau, c'est que plus les agents sont nombreux, plus les coopérations seront fréquentes et plus l'amélioration de l'ensemble de l'application est importante. Ici, cela se caractérise par une convergence plus rapide des agents vers la topologie réelle du réseau.

En fait, cette coopération directe se pose comme l'élément de base de toute phase de dialogue entre agents et servira à proposer tout un ensemble de coopérations plus complexes mais qui respecteront les critères de localisation et d'asynchronisme imposés par la coopération directe.

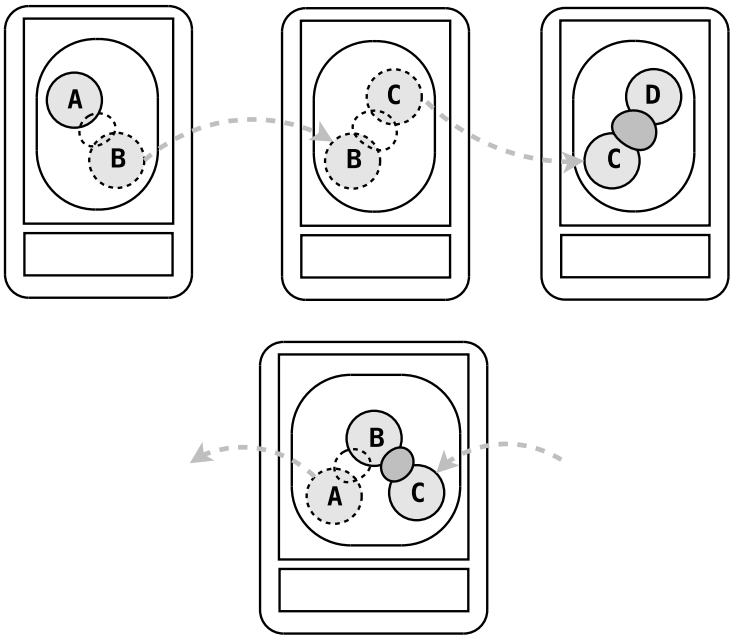


FIG. 2.4 – Coopération indirecte - autres cas

Coopération indirecte

Nous avons limité volontairement la communication directe entre agents à une interaction locale afin de pallier au problème d'accessibilité intermittente des sites dans les environnements dynamiques. Mais nous avons vu que nous souhaitions pouvoir mettre en place une communication déléguée afin d'offrir aux agents un mode d'échange plus riche (cf 2.1.4) . La coopération indirecte est proposée pour permettre ce type de communication.

Pour réaliser une coopération indirecte à distance entre deux agents, nous utilisons des agents intermédiaires qui servent, grâce à leurs déplacements, de support à la communication. En fait, il s'agit d'une suite de coopérations directes successives avec les agents intermédiaires afin de mettre en relation les participants d'une coopération indirecte. Prenons l'exemple décrit par la figure 2.3, si deux agents A et C veulent réaliser une coopération indirecte, ils utilisent le déplacement d'un agent intermédiaire B.

Pour étendre ces possibilités d'utilisation et ainsi répondre à des situations plus complexes, la coopération indirecte n'est pas limitée au cas de la figure 2.3, c'est à dire un seul agent intermédiaire réalisant deux coopérations directes successives. La coopération indirecte peut donc se définir sur plusieurs schémas allant d'une séquence de coopérations directes entre plusieurs intermédiaires jusqu'à un agent local qui attend l'arrivée de l'autre participant, comme le montre la figure 2.4.

Dans les environnements dynamiques où regrouper deux agents sur le même site représente l'un des problèmes récurrents, la coopération indirecte permet de différer en temps et/ou localisation une coopération directe qui pourrait être difficile à réaliser. Nous pensons de suite au cas où un agent n'arrive pas à rattraper un partenaire, ici l'agent cible peut laisser les informations recherchées par l'agent poursuiveur et éviter ainsi une chasse poursuite interminable.

L'une des applications les plus représentatives de l'utilité de cette coopération indirecte est la mise en place d'algorithmes basés sur la notion de rumeurs ou encore appelés *algorithmes*

des fourmis [CD98]. Dans ces algorithmes, un ensemble d'agents se déplace en collectant des informations, chacun d'eux traite les données récupérées et laisse sur les sites visités ce qu'il estime important pour les autres agents. Lorsque ceux-ci récupèrent les éléments déposés, ils les traitent à leur tour en fonction de ce qu'ils auront appris durant leurs propres migrations. Si l'information se confirme, ils accroissent sa véracité pour le reste des agents ou la diminuent dans le cas contraire.

Ces algorithmes fourmis sont mis à profit dans la découverte et la maintenance de contexte, comme celle de la topologie [TS04] où les continuels changements de l'environnement sont récupérés par ces mécanismes de rumeurs qui permettent d'adapter rapidement la perception de l'environnement, comme dans le cas de la cartographie d'un réseau après le déplacement ou la disparition d'un site. Nous verrons dans le chapitre 3 que cette méthode peut s'appliquer pour résoudre le problème de localisation des agents.

2.2.5 Autres formes d'interaction

La composition

Une caractéristique notable des agents mobiles est leur capacité d'adaptation au contexte changeant. Lors de ses déplacements, un agent collecte des informations qui lui permettent de s'adapter aux contraintes liées à l'accomplissement de sa tâche. Prenons l'exemple des politiques de sécurité, lorsqu'un agent souhaite parcourir un réseau, il est souvent confronté à des politiques intrinsèques et peut avoir à modifier son comportement en adoptant les règles du réseau à visiter, comme des communications sécurisées par exemple.

Pour permettre d'adapter son comportement, un agent peut récupérer un élément particulier qui lui permettra d'accomplir sa tâche. Cet élément peut être un simple objet, comme un certificat, mais peut tout à fait être un agent complet. C'est ce dernier cas que nous nommons par composition. Restons dans le cadre de la sécurité des réseaux, on peut imaginer que le nombre d'agents en déplacement soit limité et que le système propose un ensemble « d'agents autobus » qui véhiculeront ceux qui n'entrent pas dans les quotas.

La composition peut se réaliser lorsqu'un agent absorbe complètement un autre agent ou en utilisant des mécanismes de hiérarchisation [Sat00, Sat04b] (une arborescence) qui permettent de raisonner sur un groupe d'agents. Pour la hiérarchisation, on détermine l'agent père (origine) de la hiérarchie et on obtient la composition rattachant les agents au père. Lorsque l'agent père se déplace, c'est toute la hiérarchie qui migre avec lui et on obtient ainsi le comportement classique d'un agent composé. L'intérêt de cette méthode repose sur la facilité d'échange d'agents entre différentes hiérarchies facilitant d'autant les actions de composition/décomposition. De plus, le découplage entre l'agent père et ceux composés garantit une meilleur tolérance aux fautes, car la disparition du père n'entraîne pas automatiquement celle du reste de la hiérarchie.

La délégation

Un agent peut identifier lors de son exécution plusieurs sous tâches indépendantes qu'il pourrait traiter en parallèle. Dans ce cas, nous proposons à l'agent la possibilité de créer des sous agents auxquels il attribue les sous-tâches en question puis, optionnellement à la fin de leur exécution, il récupère les résultats obtenus. C'est ce mécanisme que nous nommons délégation.

Un cadre d'application de la délégation se porte sur les calculs diffusants souvent utilisés pour implanter des applications Internet telles que la recherche d'information ou le commerce électronique. Un calcul diffusant correspond à toutes les applications composées d'un processus initiateur effectuant un calcul local puis lançant l'exécution d'un ensemble d'autres processus possédant le même comportement.

Ce type de comportement peut facilement s'exprimer grâce aux agents mobiles à travers une succession de duplications et de migrations. Un premier agent mobile débute l'enquête en se clonant et migrant vers des sites de proximité, chaque clone poursuit à son tour le calcul en adoptant le même comportement.

2.2.6 Conclusion sur la coopération

Nous avons étudié, dans cette partie, différentes méthodes permettant d'aider le concepteur dans la construction d'une application sur des environnements dynamiques. À partir de cette étude, nous pouvons maintenant choisir quels sont les types de coopération les mieux adaptés à l'intergiciel que nous avons décrit. Pour commencer, nous éliminons d'office la coopération entre sites nécessitant une communication distante qui n'est pas possible dans l'intergiciel souhaité et qui a été présentée à titre de comparaison.

La coopération site/agent fait un pas en avant vers la description d'interactions purement locales permettant de ne pas se soucier de rupture de communication. Cependant, cette méthode repose toujours sur des environnements où les sites restent stables permettant aux agents de toujours retrouver les éléments qu'ils recherchent. Avec une accessibilité des sites intermittente, les agents peuvent perdre la trace de cet élément visé et être dans l'incapacité d'accomplir leurs tâches. C'est pour cette principale raison que nous éliminons aussi la coopération site/agent.

La coopération entre agents correspond parfaitement à l'idée que nous nous faisons d'une application répartie sur des environnements dynamiques. En effet, les agents évoluent en utilisant les sites comme un simple support d'exécution et de rencontre. Les sites s'occupent essentiellement de gérer les ressources locales et les informations système qui ne seront pas indispensables pour la réalisation des tâches applicatives des agents. De plus, toutes leurs interactions se réalisent localement en se fondant sur des méthodes de coordination qui garantissent la continuité des phases de coopération. Ce sont ces deux facteurs qui nous permettent d'adopter la coopération entre agents comme la méthode la mieux adaptée à la conception d'applications réparties dans les environnements dynamiques.

Par contre, cette méthode nécessite un investissement supplémentaire de la part du concepteur de l'application, car en plus de la tâche globale accomplie par ses agents, il devra parfaitement déterminer toutes les coopérations envisagées, accompagnées des protocoles de coordination et de la politique générale de coopération, et définir toutes leurs descriptions qui devront être fournies aux partenaires. Pour cette publication, le concepteur dispose, sur chaque site, d'un service d'annuaire local accessible d'une manière unique par tous les agents présents.

Nous retiendrons principalement que nous avons introduit une coopération entre agents complètement locale qui nous affranchit des problèmes liés aux déplacements des sites. En effet, lors des phases de coopération, le site peut se déplacer autant de fois qu'il le souhaite. Cela ne remettra pas en cause les phases de dialogue en cours. Par contre, entre le moment du début de la coopération et celui de la fin, les déplacements du site peuvent avoir modifié radicalement le contexte et pourront nécessiter une adaptation non négligeable des agents.

En y regardant de plus près, on va pouvoir distinguer deux grands types d'agents mobiles. D'un côté, nous trouverons des agents mobiles légers, dit système, qui seront assez nombreux et se déplaceront très rapidement afin de réaliser les services de base du système. De l'autre, nous auront des agents lourds, dit applicatifs, qui seront peu nombreux et se déplaceront peu fréquemment car ils réaliseront les tâches applicatives de haut niveau nécessitant des phases de traitement local beaucoup plus longues. Nous verrons dans la partie 3 plus précisément le rôle de chaque type d'agent.

En définissant les agents mobiles, leurs modes de déplacement et leurs manières de communiquer, nous obtenons un ensemble d'éléments permettant de concevoir des applications multi-agents déployées sur les intergiciels que nous avons présentés dans la partie précédente. Dans la section qui suit, nous proposons, pour ces applications, différents domaines en mettant en avant un certain nombre de contraintes auxquelles pourront répondre notre modèle d'agents mobiles coopérants qui constitue la partie II de ce document.

2.3 Domaines d'application

Dans cette partie, nous présentons quelques domaines d'applications envisageables pour les agents mobiles. Nous nous intéressons à des domaines où les capacités propres des agents mobiles pourront être mises à profit. Nous avons identifié ces principaux atouts dans la partie 1.4 : il s'agit de l'autonomie, de l'adaptabilité et des interactions locales. Ces qualités sont très utiles lorsque l'on souhaite mettre en place des explorations de réseau, représenter un utilisateur nomade ou encore lorsque les environnements sont autonomes et décentralisés [MS99], comme on peut les retrouver dans les réseaux peer-to-peer [Dun01].

2.3.1 Maintenance répartie (réseau)

Le premier domaine d'application pour les agents mobiles va s'appuyer sur leur faculté d'exploration. Nous désignons par *maintenance répartie*, la mise à jour des ressources réparties au sein d'un réseau. Ces ressources peuvent être applicatives (services) et/ou matérielles (routeur-serveur). Ces mises à jour consistent à remplacer, ou à ajouter, des fonctionnalités à un ensemble d'éléments. Par exemple, actualiser un routeur avec la nouvelle version d'un protocole.

De manière classique, la méthode de mise à jour de ressources distribuées peut s'effectuer de deux manières : soit c'est la ressource elle-même qui contacte un serveur de mise à jour, soit c'est un service de mise à jour (ou un administrateur) qui contacte toutes les ressources à actualiser. Dans les deux cas, nous sommes face à une vision centralisée où il est fait référence à un gestionnaire principal. Ces méthodes seront difficiles à adapter dans des réseaux non-structurés comme ceux construits sur le modèle *ad hoc*.

L'utilisation des agents mobiles permet de décrire simplement l'exploration d'un réseau et permet d'envisager une mise à jour totalement découplée d'un service central en déléguant une partie de l'administration aux agents mobiles. Dans ce cas, un agent possède la mise à jour à appliquer et va se déplacer de site en site, que se soit un ordinateur hôte ou un routeur, et va actualiser tous les éléments trouvés. En se déplaçant, les agents peuvent plus facilement accéder aux éléments appartenant aux infrastructures décentralisées et surtout peuvent, grâce à leur traitement local, appliquer la mise à jour sans craindre d'être interrompus [MSS+01].

En mettant en œuvre les méthodes de coopération décrites dans la partie précédente, on peut définir une exploration plus efficace en mettant en place un mécanisme d'exploration basée sur la répartition des sous-ensembles d'un réseau entre un ensemble d'agents mobiles. Par exemple, on peut imaginer qu'un groupe d'agents est créé pour mettre à jour un réseau dont la topologie est inconnue. Chaque agent parcourt le réseau à la recherche des éléments à mettre à jour tout en cartographiant le réseau en référençant les voisins des sites visités. Lorsque deux agents se rencontrent sur un site, ils coopèrent pour mettre en commun les parties du réseau découvertes et s'accorder sur la répartition des sites que chacun doit mettre à jour [MKM99].

De tels mécanismes de coopération permettent d'obtenir une maintenance de réseau bien plus efficace que les méthodes classiques basées sur un service centralisé [RDP00]. On peut aussi imaginer appliquer la méthode du *calcul diffusant* [Cub01], où lorsqu'un agent explorateur rencontre un routeur qui gère n sous-réseaux, il crée n agents auxquels il délègue l'exploration d'un des sous-réseaux identifiés. Lorsque tous les agents ont terminé, *i.e* qu'ils ont mis à jour l'intégralité du sous-réseau, l'agent explorateur passe à une autre partie du réseau.

2.3.2 Découverte de contexte

Lors de la description des architectures sans fil (cf sec. 1.1.2), nous avons vu que celles-ci peuvent s'appliquer dans des environnements ne disposant pas d'infrastructure fixe de référence. Lors d'une catastrophe naturelle, par exemple, les systèmes de communication pouvant être réduit à néant, il est intéressant de disposer rapidement de moyens pour coordonner les secours. Dans de tels domaines, où l'on ne dispose pas de gestionnaire centralisé, il est primordial de récupérer n'importe quelles informations permettant de caractériser la situation des éléments présents dans l'environnement immédiat, encore appelé *contexte*. La *découverte de contexte* se définit alors comme l'utilisation du contexte afin de donner des informations significatives aux applications et/ou des services aux utilisateurs [ADB+99]. D'un point de vue architecture, c'est le mécanisme de découverte de contexte qui permet de construire les architectures *ad hoc* et ainsi d'obtenir un réseau autonome décentralisé.

Dans la découverte de contexte, les caractéristiques des agents mobiles peuvent être mises à profit [Zas04]. L'exploration de l'environnement est facilité par la mobilité des agents qui peuvent se charger de trouver toutes les informations utiles présentes sur les sites voisins. De plus, leur capacité d'adaptation leur permet d'interpréter le contexte en fonction de besoins spécifiques et/ou de réagir en fonction d'événements perçus dans le contexte : par exemple, récupérer certains services bien précis ou capter le déplacement d'une unité mobile.

Une application envisageable de la découverte de contexte est la « localisation virtuelle » des utilisateurs [Sat04a]. En déterminant quels sont les éléments les plus proches d'un utilisateur nomade, on peut définir sa localisation et ainsi lui proposer des services adaptés [JGC04]. Par exemple, si un employé se déplace dans les différents étages de son entreprise, il pourra imprimer sur l'imprimante la plus proche.

2.3.3 Grille de calcul

Le « grid computing » consiste à regrouper dynamiquement au sein d'un même réseau virtuel, *i.e* logiciel, tout un ensemble de machines, de ressources et d'utilisateurs [BBL02]. Ceux-ci peuvent être hétérogènes et dispersés à l'échelle planétaire. L'objectif des grilles de calcul est de permettre à des organisations dispersées (entreprises, universités, etc.) de partager des ap-

plications (services), des données et des ressources (processus, disque). Le but est de mettre en commun les capacités de chaque entité. Ainsi, on peut très bien imaginer qu'une entreprise, spécialisée dans les calculs à haute performance, loue du temps de calcul sur ses serveurs à une entreprise souhaitant mettre en place une expertise précise (service). Nous sommes donc face à un environnement hétérogène, décentralisé et dynamique. Nous pouvons citer, à titre d'exemple, les réseaux peer-to-peer et le projet « *seti@home* ».

Avec les grilles de calcul, les agents mobiles trouvent un champ d'application naturel. En effet, les caractéristiques de déplacement, d'adaptation, de coopération vont permettre de tirer partie de ces environnements. On peut voir une grille comme un marché ouvert où des marchands (organisations) proposent un ensemble de produits (services) aux clients (utilisateurs) présents dans l'environnement. Les clients explorent le marché pour trouver les meilleurs produits en fonction de leurs besoins. Les agents mobiles vont pouvoir adopter ce comportement naturel de négociation. Si un client cherche à utiliser un service en fonction d'un certain nombre de critères, il crée un agent qui va aller de serveur en serveur afin de trouver le service correspondant le mieux aux critères définis par l'utilisateur. Une fois qu'il l'a trouvé, il réalise le service visé et ramène les résultats à l'utilisateur.

Le deuxième intérêt des agents mobiles est leur capacité à représenter un utilisateur déconnecté. Dans les grilles de calcul, les organisations offrent, généralement, des expertises de haut niveau nécessitant de longue phases de traitement et/ou la mise en œuvre d'un ensemble de sous-services. Avec un utilisateur nomade, se connectant de manière intermittente à la grille, l'utilisation des agents mobiles permet de déléguer la réalisation de la tâche que l'utilisateur souhaite exécuter. L'autonomie de l'agent permet à l'utilisateur de sortir de la grille durant le temps de la réalisation de la tâche. Lorsque l'utilisateur se connectera et que l'agent aura achevé sa tâche, il pourra récupérer les résultats obtenus par l'agent. L'utilisateur peut, de plus, avoir changé complètement de lieu entre temps.

Enfin, un des problèmes majeurs des grilles de calcul vient de leur construction à l'échelle planétaire. Avec une telle proportion, les communications distantes sont confrontées aux problèmes de faible taux de transferts et de fort temps de latence qui ne permettent pas d'engager des services nécessitant des communications intenses avec le client. Dans ce cas, les agents mobiles, par leur représentation locale du client, permettent de pallier à ces problèmes et offrent le niveau d'interaction nécessaire à la réalisation du service.

2.3.4 Application orientée client nomade

Nous avons vu dans la section 1.1.2 que la chute des prix du matériel de transmission sans fil, avec l'augmentation de l'autonomie des batteries et enfin avec l'augmentation des performances de la miniaturisation des processeurs de forte puissance, nous permet d'avoir des unités mobiles de petite taille, tel les assistants numériques (PDA) ou encore les smartphones, capables de se déplacer facilement avec leur propriétaire tout en exécutant des applications complexes et en ayant la capacité de rester connectées à un réseau sans fil.

Avec ces unités mobiles, les utilisateurs nomades vont vouloir accéder à leurs applications et données favorites quels que soient les environnements les entourant. D'un point de vue matériel, on peut atteindre ce besoin d'ubiquité en mettant en place les topologies hybrides que nous avons présentées dans le chapitre précédent. Par contre, d'un point de vue logiciel, il va être difficile de mettre en place une architecture type client/serveur pour suivre le déplacement des utilisateurs. En effet, les réseaux sans fil sont par définition moins performants que les

réseaux filaires et sont surtout assujettis à une faible fiabilité des liens de communication. Ces deux problèmes peuvent être contournés en utilisant les agents mobiles [VM04] qui limitent l'utilisation du réseaux en interagissant localement avec les services visés.

De plus, lors de ces déplacements, un utilisateur va se retrouver au sein d'environnements aux caractéristiques diverses et variées, que se soit au niveau de la performance du réseau ou au niveau de la multitude de services. La capacité d'adaptation des agents mobiles va leur permettre de s'accommoder de cette diversité en fonction des besoins des utilisateurs [CHK+00]. Par exemple, lorsqu'un utilisateur nomade souhaite afficher ses photos, l'agent pourra prétraiter une image haute définition pour l'adapter à la résolution de l'écran de son unité mobile. D'un côté système, l'hétérogénéité des unités mobiles, allant du PC portable jusqu'au téléphone, et celles des moyens de communication, allant du Wifi au GPRS, va demander d'adapter les services en fonction des contraintes matérielles en accord avec les critères définis par le concepteur. Ainsi, l'agent pourra s'adapter aux performances du terminal mobile en supprimant certaines fonctionnalités.

Enfin, durant ses déplacements, un utilisateur nomade peut perdre temporairement sa connexion avec un réseau, en arrivant dans une zone non-couverte par exemple. Dans ce cas, la possibilité des agents mobiles de représenter un utilisateur déconnecté permet de continuer les services engagés malgré la perte de connexion [TVP00]. On retrouve ici l'exemple d'utilisation de la grille où un utilisateur lance une simulation et vient récupérer plus tard ses résultats bien qu'il se soit déplacé entre temps.

2.4 Conclusion

Après avoir consacré le premier chapitre à la présentation des environnements dynamiques et à la mise en comparaison des agents mobiles avec les autres méthodes de programmation réparties nous avons étudié, dans ce chapitre, plus concrètement le domaine des agents mobiles en décrivant les différentes plates-formes existantes, les mécanismes d'interaction au niveau agent et enfin les domaines d'applications.

En ce qui concerne les plates-formes d'agents mobiles, nous nous sommes attachés à présenter un nombre restreint d'intergiciels car chacun d'entre eux possède une caractéristique notable pouvant être utile dans les environnements dynamiques. A partir de cette étude comparative, nous avons identifié les différents éléments de base pour avoir un intergiciel idéal et une approche de conception adéquate. Nous les résumons comme suit :

Intergiciel	Approche de conception
- L'agent est la base du système - Une communication strictement locale (hors migration) - Une communication par délégation - Une communication expressive - Une migration proactive faible - Un annuaire local de services	- Un modèle de description comportement/transition - Une assistance à la migration - Des mécanismes de synchronisation inter-agent - Un service de contexte local

A partir des éléments de l'intergiciel, nous avons étudié les différentes méthodes d'interactions qui peuvent être mises en place par le concepteur d'une application répartie. En considérant l'agent comme l'entité de référence dans le système, *i.e* les sites sont de simples supports d'exécution et de rencontre, et ne permettant que des communications locales, la seule interaction réalisable est donc la communication inter-agents que nous désignons par *coopération*. Cette coopération est complètement à la charge du concepteur qui devra définir toutes les interactions envisagées ainsi que le protocole de coordination et la politique générale de coopération. L'intérêt de cette coopération est de ne plus être exposée aux problèmes de ruptures de communication liées aux déplacements de sites et des liens versatiles dans les environnements sans fil.

Enfin, nous avons terminé ce chapitre par la présentation des différents domaines d'applications envisageables pour les agents mobiles. Un vaste domaine concerne les réseaux décentralisés dynamiques ne pouvant plus offrir une couche système qui prenne en prendre en charge les communications distantes et qui puisse offrir une représentation stable du contexte. Dans de tels contextes, les propriétés d'autonomie, d'adaptation et d'exploration des agents mobiles vont permettre d'exprimer naturellement la prise en compte du dynamisme de l'environnement. Nous avons donc proposé quatre domaines d'applications qui peuvent tirer partie de ces propriétés. Il s'agit de la maintenance répartie, de la découverte de contexte, des grilles de calcul et des applications associées à des clients nomades.

Dans cette première partie de la thèse, à savoir l'état de l'art général des applications distribuées suivi de celui plus spécifique aux agents mobiles, nous avons mis en évidence que les environnements à venir sont de plus en plus dynamiques, de moins en moins centralisés et à large échelle (planétaire). Cette caractéristique va imposer une nouvelle manière de concevoir les applications réparties. En effet, nous sommes confrontés à une double mobilité qui ne peut être prise totalement en compte par le système, le rendant incapable d'offrir une représentation stable et globale de l'environnement. Cette double mobilité vient du déplacement des unités mobiles, nous l'appelons *mobilité matérielle*, et de la mobilité des agents, nous la désignons par *mobilité logicielle*.

Dans la partie suivante, nous décrivons plus précisément le problème de la double mobilité en mettant en exergue le manque de moyens pour la prendre en compte, puis nous proposons une nouvelle approche pour résoudre ce problème et enfin nous exposons notre modèle d'agents mobiles coopérants permettant de prendre en compte les environnements dynamiques à large échelle.

Deuxième partie

Agents Mobiles Coopérants

Gérer la double mobilité

Thèse soutenue

3

Dans le précédent chapitre, nous avons fait un état de l'art qui porte sur les plates-formes supportant le modèle des agents mobiles, sur les différentes méthodes de communication utilisables pour la conception d'applications multi-agents ainsi que sur leurs domaines d'application. Nous avons aussi présenté dans le premier chapitre que les architectures réseaux tendent à devenir hybrides et vont induire des environnements de plus en plus dynamiques.

A partir de ces deux éléments, logiciel et matériel, nous commençons ce chapitre par détailler plus précisément, indépendemment de l'intergiciel, les caractéristiques propres d'un déploiement d'applications formées d'agents mobiles s'exécutant sur des sites eux aussi en mouvement. Nous verrons que le problème de localisation des agents, soulevé dans le précédent chapitre, s'amplifie à cause de cette double mobilité.

Nous verrons, dans un deuxième temps, que nous envisageons l'utilisation des applications multi-agents dans un cadre à large échelle, que ce soit en nombre d'agents ou de sites, imposant la prise en compte de certaines particularités. Ensuite, nous présentons notre solution permettant de faire face à ces deux paramètres de mobilité et de large échelle.

3.1 La double mobilité

Le succès récent des technologies sans fil, que ce soit grâce au téléphone cellulaire ou au réseaux Wifi, a induit de nouveaux types d'architecture dites hybrides (cf 1.1.3). Celles-ci apportent l'ubiquité de communication aux utilisateurs nomades car ils peuvent accéder à internet dès qu'ils le souhaitent. Cependant la mobilité physique rend les environnements fortement dynamiques et pose la question essentielle de l'adressage physique. Nous commencerons l'étude de la double mobilité par ce point.

En partant de ces constatations de la mobilité croissante des sites et de l'ubiquité apportée aux utilisateurs, nous affirmons que les agents mobiles proposent le modèle le mieux adapté à la prise en compte de ces environnement dynamiques, ceci grâce à leur autonomie et leur mobilité. Cette utilisation des agents mobiles va alors induire un deuxième niveau de mobilité, celui logicielle, que nous abordons dans un deuxième temps.

3.1.1 Mobilité physique

Lorsqu'un élément souhaite rejoindre un réseau déjà établi, il va commencer par se connecter aux autres membres via une liaison filaire ou « par onde », puis il obtient, généralement grâce à un gestionnaire, une adresse unique permettant de l'identifier sans ambiguïté au sein de ce réseau. Il peut enfin commencer à communiquer avec les autres membres du réseau. Notons que l'adresse peut être attribuée de manière statique lors de l'installation du système ou dynamiquement lors de la connexion.

Cette méthode trouve ses limites dès lors que nous passons dans un environnement dynamique car elle ne pourra pas gérer le nombre important de connexions/déconnexions. La gestion des adresses attribuées va se heurter au problème du nombre croissant d'unités mobiles et surtout la faible bande passante disponible dans les topologies hybrides va limiter les performance de communication entre les divers éléments du réseau.

Les limites matérielles

Pour le moment, les extensions du réseau, grâce aux technologies sans fil, s'effectuent à travers la mise en place de bornes d'accès auxquelles se connectent les utilisateurs nomades, ce qui est communément appelé réseaux à station de base. Cette technique est limitée par les propriétés même des points d'accès construisant le réseau. Nous pouvons en extraire deux principales restrictions.

Premièrement, les bornes sont limitées par le nombre de connexions qu'elles peuvent gérer en simultané et par conséquent limitent le nombre d'utilisateurs nomades envisageables. Les solutions couramment utilisées passent par la multiplication du nombre de points d'accès permettant logiquement d'augmenter le nombre possible de connexions, c'est la méthode utilisée par les réseaux de télécommunication. Le problème de cette technique vient surtout du coût engendré par la multiplication des bornes imposant une couverture orientée vers les zones d'une forte densité d'utilisateurs, comme les halls de gare, au détriment des zones faiblement fréquentées, comme les espaces ruraux, ou difficiles d'accès comme les parkings souterrains.

Deuxièmement, les bornes d'accès ont à disposition une bande passante maximale théorique qu'elles partagent entre tous les clients en cours de connexion. Cette caractéristique peut faire chuter considérablement les performances du réseaux, ce qui est difficilement acceptable pour des utilisateurs de plus en plus exigeants sur ce point, surtout depuis l'arrivée des connexions haut-débit à domicile. Ce qui est regrettable dans ce cas, c'est la sous exploitation des capacités de transmission propres aux unité mobiles car elles dépendent de la connexion établie avec la station d'accès.

Ces deux éléments, illustrés sur la figure 3.1, nous poussent à conclure que les topologies hybrides vont tendre à devenir la référence. En donnant la possibilité, grâce aux points d'accès, d'accéder à Internet tout en permettant une extension basée sur les relais des noeuds eux-mêmes, ces réseaux éliminent les problèmes liés à la limitation du nombre de connexions, réduisent les coûts du déploiement physique et permettent de conserver une meilleure performance de la bande passante en répartissant le trafic général sur l'ensemble des noeuds. La réduction des coûts de déploiement permettant éventuellement de mieux répartir géographiquement les bornes en se focalisant moins sur la densité utilisateur, ceci permettant peut-être de couvrir les zone difficiles d'accès.

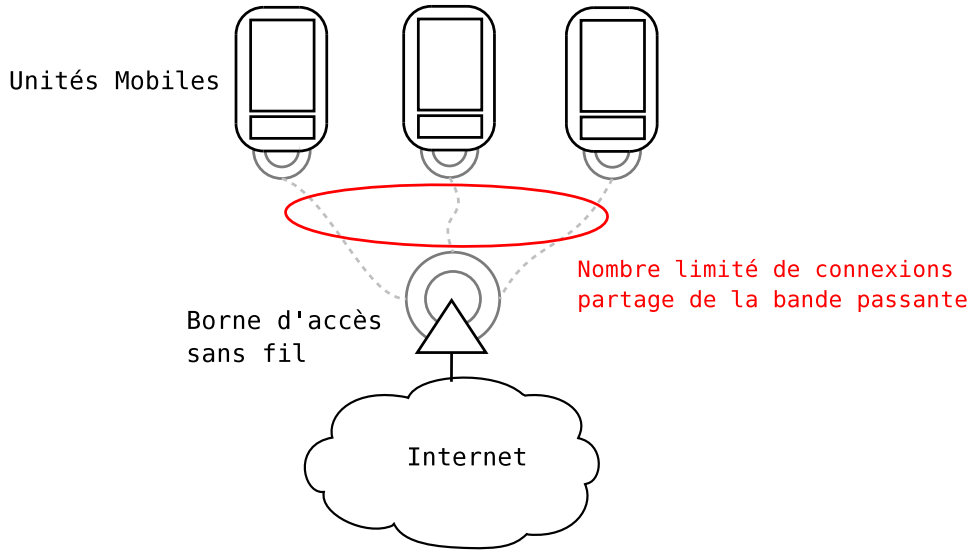


FIG. 3.1 – Limites matérielles

Avec l'utilisation des noeuds mobiles comme relais, nous obtenons des environnements dont l'infrastructure est en perpétuelle évolution et nécessite une reconfiguration constante. Celle-ci est généralement à la charge de la couche réseau mais nous allons voir que les méthodes existantes ne peuvent par faire face à ces changements incessants au niveau matériel.

Les limites logicielles

Le protocole d'identification et de communication le plus répandu sur les réseaux est sans conteste IP qui permet d'affecter une adresse unique à une unité, de trouver et dialoguer avec tout élément rattaché au réseau. Sur les topologies hybrides, plusieurs projets tentent de permettre une utilisation des adresses IP afin de rendre compatible les unités mobiles avec les infrastructures déjà en place et surtout afin de rendre transparent leurs déplacements, nous allons en présenter deux.

Pour remédier au nombre croissant de machines inter-connectées grâce au protocole IP, l'évolution de la norme à travers IPv6 [Ciz98] permet principalement d'avoir un ensemble d'adresses suffisant permettant d'attribuer une adresse unique à toutes ces machines dont le nombre est en expansion constante. Cette norme a proposé de traiter d'emblée le déplacement d'une unité en intégrant le principe MOBILE-IP qui fonctionne selon le modèle des «liens de poursuite» [DRL98]. Chaque unité mobile détient une adresse IPv6 principale, référencée dans son réseau d'origine. Lors d'un déplacement, elle obtient une adresse temporaire, qu'elle communique au routeur de son sous-réseau d'origine qui agira comme son représentant et servira de proxy. Il intercepte les paquets lui étant destinés et crée un tunnel entre l'unité visée et l'émetteur des paquets. Le déplacement reste invisible pour l'unité en déplacement ainsi que pour ceux qui souhaitent communiquer avec elle.

Nous constatons que MOBILE-IP possède deux caractéristiques pouvant être difficilement associées à des unités fortement mobiles. La première est le maintien de la liaison entre l'adresse principale et l'adresse temporaire qui peut augmenter de manière significative le trafic dans des réseaux sans fil avares en bande passante. La deuxième est l'obligation de posséder un adresse de base, alors que rien n'oblige un possesseur d'unité mobile d'avoir un fournisseur d'accès

personnel pour utiliser les réseaux sans fil publics. Nous voyons que MOBILE-IP apporte une première gestion des déplacements, par exemple lorsqu'une entreprise envoie un employé en mission chez un de ses clients. Cependant cette technique reste inadaptée au fort dynamisme des sites que nous envisageons.

Le groupe de travail MANET de l'IETF s'attache à standardiser le protocole de routage IP dans les réseaux «Mobile Ad'hoc NETwork» Ce groupe cherche à rendre transparent l'utilisation de IP dans des réseaux fortement dynamiques où chaque noeud est un routeur. Il n'est pas nécessaire ici qu'un élément mobile du réseau possède une adresse principale, elle pourra lui être attribuée dynamiquement. Le groupe MANET nous permet de considérer que la communication inter-machines peut se réaliser dans les topologies dynamiques grâce à la couche réseau qui prend en charge le routage.

Nous supposons donc que nos applications évoluent dans un environnement où les sites changent dynamiquement d'adresses et que le réseau prend uniquement en charge le routage des paquets. Cependant, actuellement la couche réseau ne peut pas prendre en charge la reconfiguration continuelle de l'infrastructure engendrée par les fréquents changements d'adresses des sites et pas conséquent le routage général entre tout point d'une architecture hybride n'est pas assuré. Nous nous plaçons donc dans le cas où nous ne pouvons pas faire communiquer n'importe quels éléments de l'architecture mais seulement ceux qui sont proches.

De plus, ces continuels changements d'adresses vont poser un autre problème pour un service rendu habituellement par le système : la résolution de nom. Nous savons pertinemment que lors de la conception d'applications réparties, les développeurs utilisent rarement les adresses des sites directement, mais utilisent plutôt leurs noms symboliques, dont la concordance avec l'adresse physique est gérée par le service de nommage. Celui-ci utilise une structure hiérarchique et n'est pas conçu pour des mises à jours dynamiques trop fréquentes. Par conséquent nous devons aussi considérer que le système ne prendra pas en charge la résolution des noms des éléments nomades.

Donc, avec ces limites matérielles et logicielles, nous ne devons plus considérer que le système support des applications va prendre en compte les changement de l'infrastructure et nous devons admettre qu'il n'offre pas une représentation stable de l'environnement. Ce critère de dynamisme devra être pris en compte lors de la conception des applications.

Raisonner localement

Nous avons vu dans la section 2.2 que la localisation des agents au sein du système est définie par le site support en cours et que nous pouvions avoir des migrations ciblées indiquant précisément la destination que l'agent souhaite atteindre. Cependant nous venons de voir qu'il n'était pas envisageable d'utiliser les habituelles techniques de routage général et de résolution globale de noms. Pourtant nous avons besoin que le système permette aux agents de déterminer où ils souhaitent aller et qu'il leur offre le moyen de se déplacer.

Pour trouver de nouvelles méthodes système, nous nous orientons vers les techniques employées dans les réseaux *ad hoc* que nous avons présentés dans la section 1.1.2. Dans ces infrastructures dynamiques par essence, des mécanismes permettant la découverte des éléments du contexte sont intégrés afin de mettre en place dynamiquement un réseau [TWB03]. Ainsi chaque site pourra savoir quels sont les éléments atteignables et proposera aux applications un ensemble de voisins avec lesquels ils pourront communiquer. Pour éviter les conflits, chaque site possède un couple formé de son nom et du numéro de série unique attaché au périphé-

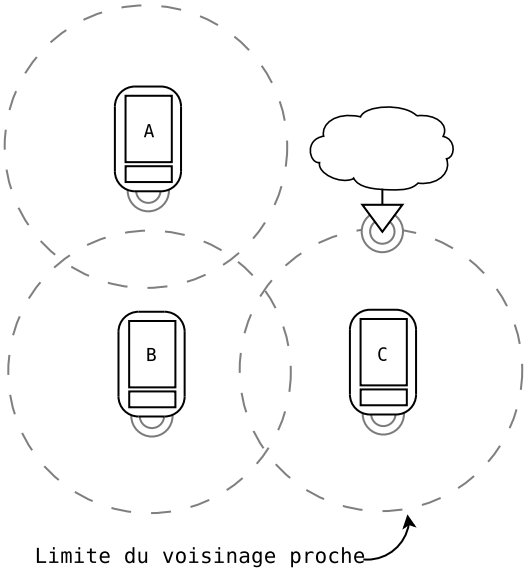


FIG. 3.2 – Voisinage proche[7]

rique de communication sans fil, ce couple permettant d'éliminer toute ambiguïté au sein d'un même réseau.

Cette technique s'accorde parfaitement avec les problèmes que nous avons soulevés, car elle permet de raisonner en terme de voisinage direct et non pas en terme de globalité, ce qui nous posait problème. Dans ce cas, chaque site va gérer son environnement direct en cherchant les éléments dans son proche voisinage, ces éléments pouvant être nomades ou fixes (comme une borne d'accès), puis il va maintenir à jour l'ensemble trouvé en renouvelant ses recherches ou en étant informé par les éléments déjà référencés, lorsqu'un site s'éteint par exemple.

Avec cet ensemble de voisins atteignables, les agents sont localisés sans équivoque grâce à l'identifiant du site et peuvent obtenir un liste de voisins vers lesquels ils pourront se déplacer. Ce sont les deux points que nous voulions obtenir. Par contre, nous voyons encore ici que le système ne prendra pas en charge le dynamisme global et que ce sont les applications qui devront l'intégrer dans leur conception. Dans l'exemple de la figure 3.2, si un agent est sur le site A, il ne disposera que d'un accès vers le site B et s'il souhaite rejoindre le reste du réseau global, il devra migrer successivement sur B puis sur C. Il faut remarquer que c'est l'agent lui même qui devra savoir le chemin à emprunter car le système ne lui fournira que des informations partielles, ici le site A ne connaît pas le moyen de rejoindre le reste du réseau car il n'est pas dans son voisinage. On peut d'ores et déjà se demander si les agents arriveront à se trouver dans un espace global géré uniquement de proche en proche. Nous étudions ce problème dans la chapitre 6.

3.1.2 Mobilité applicative

En utilisant le paradigme des agents mobiles, on introduit par définition une mobilité applicative induite par le déplacement des agents. Nous avons défini, dans le section 1.2.4, les

[7]Nous prenons comme convention que le cercle en pointillé représente la limite de « connexion ». Cela veut dire que si deux cercles se touches, les unités peuvent établir une connexion directe.

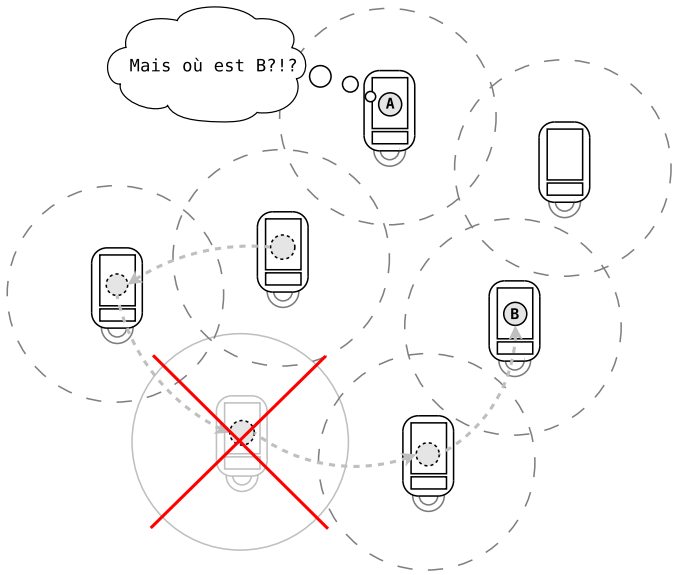


FIG. 3.3 – Problème de localisation

applications comme un ensemble d'agents mobiles cherchant à utiliser les services répartis et en dialoguant entre eux pour connaître l'avancement général de la tâche globale. Les services étant des applications comme les autres et les phases de dialogue se réalisant grâce aux coopérations locales, l'un des problèmes récurrents, que devra résoudre un agent, est de trouver les autres partenaires souhaitant participer aux coopérations.

Dans les environnements dynamiques, cette recherche va se scinder en deux principales difficultés : déterminer sur quel site se trouve l'agent recherché et trouver le moyen d'atteindre le site en question. En effet, le système ne gérant que le contexte local d'un site, il proposera uniquement l'ensemble des agents présents localement et les destinations atteignables dans le voisinage, ceci ne donnant pas le moyen suffisant aux agents d'estimer la localisation et/ou le chemin permettant d'atteindre un partenaire qui se trouverait sur un site extérieur au contexte local.

La première difficulté portant sur la localisation est due à la mobilité propre des agents. Lorsqu'ils se déplacent de sites en sites, ils ne disposent pas, à cause de la gestion locale, d'un service général de diffusion de leurs localisations successives permettant d'informer tous les partenaires répartis sur l'ensemble du système. Nous pourrions utiliser la méthode de lien de poursuite, grâce à la coopération indirecte, qui permettrait de suivre un à un les déplacements d'un agent. Cependant, les sites pouvant disparaître, un élément de la chaîne pourrait ne plus être accessible en rendant l'ensemble du lien inutilisable, comme le montre la figure 3.3. Il va donc falloir imaginer de nouvelles méthodes permettant de connaître le contexte général de localisation des agents et surtout qu'elles soient capables de réagir rapidement à leurs déplacements.

La deuxième difficulté portant sur la détermination du chemin, qui permet de rejoindre le site souhaité, est due au dynamisme de l'infrastructure auquel devront faire face les agents. En effet, une fois la localisation du partenaire déterminée, l'agent devra établir un chemin permettant de l'atteindre avec comme seule information celle des voisins atteignables depuis sa position actuelle. Mais l'agent sera confronté, en plus, aux mouvements des sites qui pourront remettre en cause le parcours établi. On imagine le cas où un site se trouve dans une partie

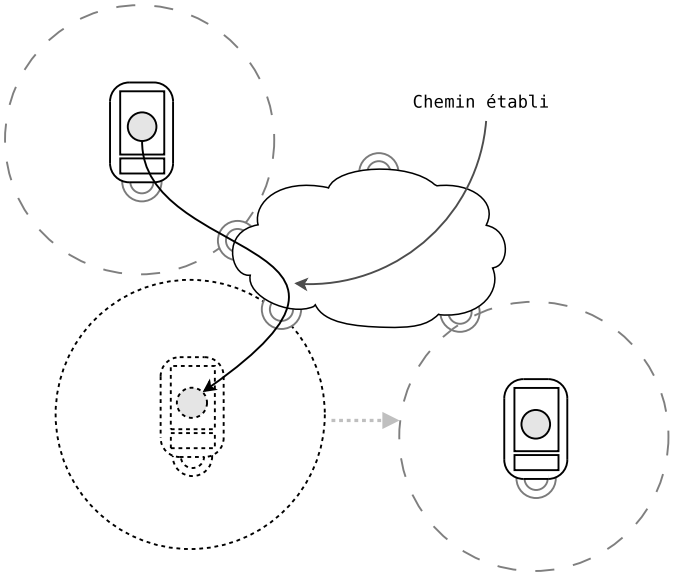


FIG. 3.4 – Problème de chemin

d'un réseau, qu'un agent se déplace vers lui et, avant son arrivée, le site se déplace vers un autre endroit, comme le montre la figure 3.4. Donc, en plus des méthodes de localisation, nous devons imaginer des moyens permettant de guider efficacement les déplacements des agents et, là aussi, ils devront s'adapter rapidement aux mouvements des sites.

Nous devons ajouter, de plus, que ces deux parties de la mobilité applicative ne sont pas strictement scindées et qu'au contraire elles interagissent l'une sur l'autre en augmentant d'autant la complexité de parvenir à établir une rencontre. Lorsqu'un agent se déplace afin de suivre le chemin le menant à son partenaire, sa localisation change régulièrement au gré de ses migrations, ceci rend encore plus difficile la détermination de sa position pour les autres agents qui cherchent à le rencontrer. D'autre part, lorsque la localisation d'un agent change, la détermination du chemin est de fait remise en cause. On voit bien que la localisation et l'utilisation du chemin s'influencent.

Nous voyons ici toute la difficulté que doit affronter le concepteur d'une application répartie car il devra prendre en charge la gestion d'un ensemble de problèmes qui sont habituellement traités par le système. En plus de la description de sa tâche globale, il devra prendre en compte les problèmes liées aux mobilités matérielles et logicielles qui ne peuvent être confiés au système, celui-ci se limitant à une gestion de son environnement local et ne peut pas proposer une couche prenant en charge l'intégralité du dynamisme.

Par contre nous pouvons noter que cette complexité va surtout gêner les applications utilisant la migration ciblée où les agents savent parfaitement les services à utiliser pour accomplir leurs tâches et cherchent explicitement à établir des rencontres. Dans le cadre des migrations libres, les agents sont peu affectés par le dynamisme car ils opèrent leurs déplacements sur des critères n'ayant pas de rapport avec les autres agents et se libèrent ainsi des problèmes liés aux déterminations de la localisation et du chemin. Avec ce type de migration, les agents vont naturellement intégrer la modification du contexte et vont se déplacer uniquement sur des critères purement personnels généralement orientés « système », comme dans le cadre de la cartographie d'un réseau, que nous avons déjà vu, où les agents iront vers les sites du voisinage qu'ils n'ont pas encore parcourus.

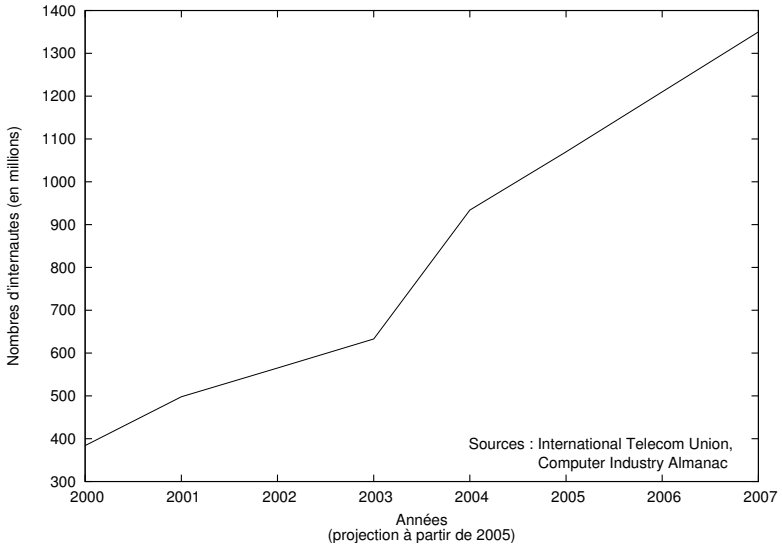


FIG. 3.5 – Évolution du nombre d'internautes

3.2 Une large échelle

Le problème de la mobilité que nous venons de présenter prendra toute sa mesure si nous envisageons nos environnement à une échelle bien plus large que celle envisagée habituellement. En effet, la gestion actuelle de la répartition sait s'accommoder d'un minimum de dynamisme en utilisant des mécanismes de résistance aux fautes qui permettent de pallier à la défection d'un site, comme dans le cadre d'un cluster de machines par exemple. Généralement ces méthodes sont bien adaptées pour résister à des montées en charge d'applications serveurs, tel les moteurs de recherche, en répartissant sur un ensemble de sites parfaitement délimités les différentes requêtes à satisfaire et en s'adaptant à des problèmes préalablement anticipés, comme la surcharge d'un élément de l'ensemble.

Mais le succès grandissant d'internet qui n'est plus à démontrer, il suffit de regarder l'évolution du nombre d'internautes représentée par le graphique de la figure 3.5 pour s'en convaincre, va demander de repenser l'échelle de diffusion de nos applications car l'augmentation continue que nous constatons va s'accompagner de changements radicaux, aussi bien niveau matériel qu'au niveau applicatif, imposant de proposer d'autres méthodes que celles employées à l'heure actuelle.

3.2.1 L'échelle matérielle

La croissance du nombres d'internautes de la figure 3.5, va s'accompagner obligatoirement d'une augmentation de machines permettant l'accès à internet mais aussi des services qui leur sont proposés. Ces deux points vont induire une recrudescence du nombre de sites sur Internet que nous pouvons voir sur la figure 3.6. Cette courbe donne uniquement les sites référencés en Europe, *i.e* possédant une entrée dans une service de nommage, et ne prend pas en compte l'intégralité des machines réellement connectées, comme celles des réseaux domestiques par exemple. Cependant, cette courbe nous donne un bon référentiel pour constater que durant ces dix dernières années (1995-2005) nous avons eu une augmentation considérable des sites avoisinant les 1700%. Ceci témoigne de l'engouement extraordinaire sucisté par Internet et son

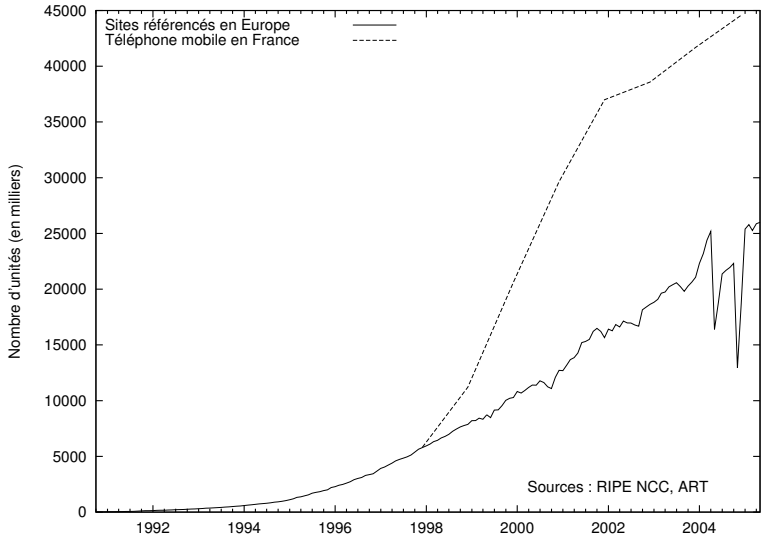


FIG. 3.6 – Évolution matérielle

entrée dans les moeurs en tant que nouveau média de masse. Mais ce succès nous oblige à reconsidérer les échelles de déploiement des applications réparties qui ne sont plus comparables avec les situations connues depuis le début d'Internet.

Un autre phénomène doit être mis en avant pour bien comprendre la taille des échelles auxquelles nous sommes à présent confrontés. C'est celui de l'informatique nomade caractérisée par l'explosion du marché de la téléphonie cellulaire. En regardant toujours le même graphique, on constate que le nombre de téléphones en France représente quasiment le double des sites référencés en Europe. Alors, certes, pour le moment, une grande majorité de ces appareils ne savent que téléphoner mais ils sont de plus en plus partie intégrante d'internet en accédant aux technologies haut débit comme l'UMTS et deviennent assez puissants pour pouvoir exécuter de véritables applications métiers.

Ces deux évolutions matérielles marquent en fait une modification du comportement des internautes qui ne sont plus liés à une machine déterminée pour accéder à Internet et par conséquent à leurs applications préférées. Jusqu'à présent le comportement général des utilisateurs se caractérisait par deux grands types accès à internet, à domicile et au travail, avec deux identités virtuelles distinctes. Mais avec la convergence des technologies mobiles, il est de plus en plus fréquent que cette distinction disparaisse. En effet, avec l'augmentation des capacités des appareils mobiles, que se soit en terme de traitement ou de capacité de stockage, un utilisateur peut maintenant se déplacer avec ses propres données, personnelles et/ou professionnelles, et les utiliser sur différents appareils et pas forcément avec des identités différentes.

Cette nouvelle approche induit un nouveau type de comportement où l'internaute est connecté quasiment en permanence avec les mêmes données mais en passant d'un appareil à un autre et/ou d'une localisation à une autre. Par exemple, en partant du travail, l'utilisateur utilise son téléphone mobile le temps de se retrouver sur son réseau domestique. Ce phénomène, d'ubiquité et de mobilité à grande échelle, ne peut être pris en compte correctement à l'heure actuel à cause des limites matérielles et logicielles que nous avons identifiées pour la double mobilité. Nous avons donc besoin de proposer des mécanismes permettant de s'adap-

ter facilement à cette large échelle matérielle et aux déplacements des utilisateurs, faculté que nous retrouvons chez les agents mobiles.

3.2.2 Un nombre important d'agents

Dans le modèle classique des applications multi-agents, l'effort principal de conception se porte sur la détermination précise du nombre nécessaire d'agents et sur l'organisation de leur répartition dans l'environnement afin d'optimiser le déroulement de la tâche générale. D'ordinaire, dans cette approche, le nombre d'agents est volontairement restreint et stable durant le cycle de vie de l'application permettant ainsi de parfaitement déterminer l'organisation optimale.

Mais comme nous l'avons déjà détaillé (cf section 2.2.1), cette conception se place généralement dans des environnements fixes où les interactions directes entre agents se limitent à ceux de la même application. Cependant, elle ne peut s'appliquer au dynamisme imposé par les topologies hybrides remettant continuellement en cause l'organisation établie et nécessitant un mécanisme de communication indirecte entre agents d'applications diverses, comme dans le cas de la coopération indirecte. Dans ce cas de figure, nous devons abandonner l'idée d'établir une organisation prédéfinie et laisser les agents s'adapter aux conditions qu'ils vont rencontrer durant leurs déplacements.

Dans notre vision des applications réparties basées sur notre modèle des agents mobiles coopérants, nous pouvons parfaitement avoir un nombre dynamique d'éléments intervenant pour une même application, qu'ils soient parties prenantes ou simples intermédiaires. Si nous prenons le cas de la coopération indirecte, plusieurs intermédiaires peuvent intervenir pour permettre de mettre en relation les deux participants finaux (cf figure 2.4 de la section 2.2), ceux-ci pouvant être des agents propres de l'application, créés pour l'occasion ou bien encore complètement extérieurs en servant juste de vecteur pour la coopération indirecte initiée par les participants.

En laissant la possibilité aux concepteurs d'utiliser et/ou de créer des agents dynamiquement durant l'exécution de ses applications, nous permettons une évolution de leur taille et, par conséquent, de ne plus la limiter à celle de l'initialisation. Avec cette échelle changeante, nous ne pouvons plus penser l'organisation dès la conception d'une manière figée mais nous devons proposer des mécanismes de rencontre et de coopération qui permettent de simplifier et d'optimiser la conception des applications.

La flexibilité de la taille des applications se justifie par la modification des comportements des internautes que nous avons décrite dans l'évolution matérielle. La future ubiquité des utilisateurs, grâce aux machines nomades, pousse de plus en plus les développeurs à proposer la possibilité d'utiliser à distance leurs applications grâce à Internet et ce, quelle que soit la localisation de l'utilisateur. Cette augmentation du nombre d'applications accessibles à distance va accroître le nombre d'agents et donner des environnements composés d'un ensemble d'éléments provenant d'horizons divers, une sorte de melting-pot logiciel de très grande taille en évolution constante. Les concepteurs pourront alors utiliser les agents présents, ou en créer d'autres, en fonction des leurs besoins pour accomplir la tâche de leur application.

Pour conclure sur cette partie, nous pouvons dire que l'augmentation des internautes, couplé à l'augmentation des éléments sur internet (mobile ou non) ainsi que l'accroissement des applications évoluant à travers la toile vont imposer un passage à une échelle, matérielle et logicielle, bien supérieure à ce que nous pouvons trouver habituellement lors des conceptions de

programmes. Pour permettre d'appréhender cette évolution sur la taille des environnements et des applications, nous devons proposer des mécanismes permettant de simplifier et d'optimiser ce nouveau type de conception du logiciel.

3.3 Les différents types d'agents

Avant de proposer des mécanismes de base pour la conception d'application répartie sur les environnements dynamiques, nous allons commencer par classifier les grands types d'agents mobiles qui peuvent être envisagés et, par conséquent, présents dans les systèmes multi-agents. Cette classification vient de notre expérience sur le déploiement des agents mobiles et n'a pas vocation à imposer des types figés mais plutôt de donner un référentiel aux concepteurs.

Pour établir une différenciation entre les agents, nous avons utilisé un certain nombre de critères. Ces propriétés, empruntées à la classification des processus systèmes, nous permettent de distinguer les différents types d'agents. En voici une liste non exhaustive :

- **Mode de migration** : déplacement ciblé ou libre par rapport aux autres agents.
- **Mode de coopération** : courte ou longue, localement nombreuses, parallèle ou séquentielle etc.
- **Identification des partenaires** : les partenaires sont connus d'avance ou choisis au hasard.
- **Volume de l'agent** : taille totale comprenant le code et données transportées.
- **Nombre de partenaires** : le nombre d'agents composant l'application d'appartenance. Dynamique ou fixe, grand ou petit.

Avec ces critères, nous référençons en fait deux grands types d'agents : les légers et les lourds. Les légers correspondent à des agents dont la tâche principale ne nécessite pas d'engager de longues phases de traitements locaux. Au contraire ils stationnent très peu de temps sur le même site en privilégiant les déplacements fréquents et rapides. En opposition, les agents lourds vont nécessiter des temps de visite sur les sites beaucoup plus conséquent à cause de longues phases de traitements locaux, ils vont dès lors se déplacer peu souvent et relativement lentement.

Nous insistons sur le fait que ces deux grandes catégories nous servent principalement à donner un référentiel au concepteur mais qu'elle ne sont absolument pas exclusives. En réalité il existe une continuité dans les comportements et la taille permettant d'aller des agents légers jusqu'aux agents lourds. Il est tout à fait envisageable de définir des agents intermédiaires qui se déplacent peu souvent mais très rapidement, comme il est envisageable de définir des services pouvant être construits sur des agents lourds et des agents légers.

3.3.1 Les agents légers

Comme nous venons de le dire, les agents légers exécutent une tâche qui effectue de courtes phases de calcul local et ils vont donc migrer fréquemment et rapidement. Nous les nommons « légers » car il s'agit d'agents de petite taille dans le sens où leur code exécutable est réduit le plus possible et les informations qu'ils véhiculent sont peu volumineuses. Cette petite taille

va leur donner la faculté d'un déplacement très bref dû à un temps de transmission très court grâce à leur faible coût en bande passante. Cette caractéristique est fort appréciable dans les environnements dynamiques visés où le lien entre deux sites a une durée de vie limitée et souvent réalisé par une connexion sans fil peu généreuse en bande passante.

L'intérêt principal des agents légers est qu'ils vont se déplacer bien plus vite que ne peuvent le faire les sites supports. En étant plus rapides à bouger que les sites à renouveler leurs voisinages, ces agents légers pourront migrer sur n'importe quel élément accessible avant que celui-ci ne disparaisse, permettant ainsi aux agents légers de ne pas réellement se soucier du problème liés aux fréquents déplacements des sites. Notons que cette rapidité de mouvement n'empêche absolument pas les phases de dialogue mais que la politique générale de migration va privilégier les déplacements plus que les longs échanges. Pour pallier à ces brèves phases d'échange, nous allons les multiplier en augmentant le nombre d'agents légers provenant d'une même application.

Le domaine d'application des agents légers s'inscrit principalement dans la réalisation des services de base du système qui demande une forte indépendance par rapport aux déplacements des sites et une grande réactivité aux changements de l'environnement. Pour ce faire, ces agents seront nombreux et vont généralement utiliser la migration libre en multipliant de brèves phases de coopération, directe ou indirecte, avec des partenaires pris au hasard. Leur tâche principale est généralement inspirée des algorithmes de rumeurs, de types fourmis [CD98], car ils correspondent parfaitement à la philosophie d'indépendance que nous recherchons vis à vis de l'environnement et des autres composants de l'application.

3.3.2 Les agents lourds

A l'opposé des légers, les agents lourds réalisent une tâche imposant de longues phases de traitements locaux et en conséquence ils effectuent de rares déplacements qui sont relativement lents. Ces agents sont dits « lourds » car la taille du code exécutable ainsi que celle des données transportées seront beaucoup plus volumineuses que celles des agents légers. Cette grande taille va imposer de lents déplacements à cause de leur grande consommation en bande passante nécessitant un long temps de transmission. Cette propriété va poser un problème dans les environnements dynamiques qui ne peuvent pas garantir la continuité des liens de communication et ne peuvent proposer qu'une faible bande passante.

Ces agents lourds sont généralement utilisés lors de la conception d'applications dites « métier » qui nécessitent un savoir faire particulier bien plus complexe que celui des agents légers. Dans ce type d'applications, le développeur connaît généralement les autres services métiers qu'il souhaite utiliser et va donc choisir de mettre en place la migration ciblée pour établir les rencontres nécessaires à la mise en œuvre de la coopération.

C'est là où ce situe le principal problème d'un agent lourd car les longues phases de traitements vont le rendre beaucoup moins mobile que les sites supports et vont nécessiter des adaptations à chaque fin de calcul pour appréhender les modifications de l'environnement et pouvoir effectuer la prochaine rencontre. De plus, vu le coût important d'un migration pour un agent lourd, il faudra minimiser le nombre de déplacements qui lui permettront d'atteindre son partenaire et par conséquent il ne pourra pas naviguer au hasard de sites en sites mais il devra orienter ses mouvements afin d'optimiser le coût général de cette phase de recherche.

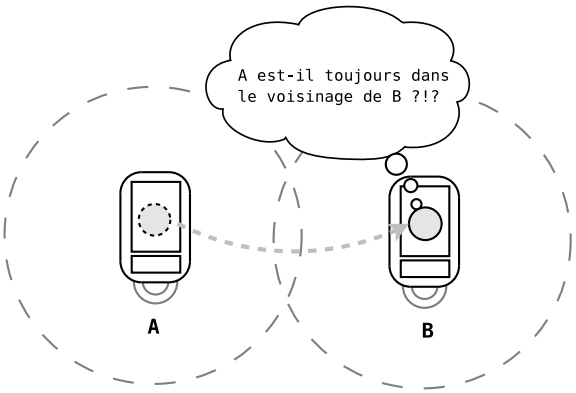


FIG. 3.7 – Perception des modifications du contexte

3.4 Une couche adaptable

En faisant une première analyse des deux grands types d'agent que nous venons d'introduire, nous pouvons constater que ces deux catégories tendent vers une certaine complémentarité. Notre principal problème vient du dynamisme inhabituelle des environnements que devront affronter les concepteurs des applications métiers en se fondant sur des agents qui utilisent la migration ciblée qui se heurte aux difficultés de localisation et de chemins imposées par la double mobilité. Or les agents légers étant mieux adaptés aux modifications du contexte, ils doivent pouvoir rendre certains services qui permettront aux agents d'obtenir une représentation simplifiée de l'environnement.

3.4.1 Le masquage du dynamisme physique

La propriété des agents légers à se déplacer plus rapidement que les sites supports leur permet de ne pas être affectés par les modifications physiques qu'ils pourront constater quasiment immédiatement suite à leurs mouvements. Prenons l'exemple de la figure 3.7, si un agent se déplace entre les sites A et B, il pourra vérifier dès son arrivée si le site A fait toujours partie du voisinage de B où s'il a quitté son voisinage. En procédant ainsi sur un ensemble réduit de sites, un agent léger pourra percevoir rapidement les modifications de l'environnement et avoir une vision courante de son contexte local très proche de la réalité physique.

En créant un ensemble d'agents légers dont la taille est correctement adaptée à celle du système et en additionnant les visions locales de chacun, on obtient une représentation générale du contexte qui se rapproche au plus près de la configuration réelle en cours. En utilisant cette ensemble comme une couche système, que nous baptisons ***couche d'ambiance***, nous pouvons alors absorber une grande partie du dynamisme matériel qui pose problème et obtenir une représentation du contexte, proche de la réalité, exploitable par les agents lourds. Notons que ce que nous venons de décrire pour le dynamisme matériel peut parfaitement s'adapter à celui du logiciel et ainsi prendre en compte la double mobilité.

3.4.2 Une couche plus stable pour les agents lourds

Nous avons vu que les agents lourds, de par leur coût élevé de migration, vont devoir minimiser le plus possible le nombre de déplacements nécessaires pour atteindre leurs partenaires.

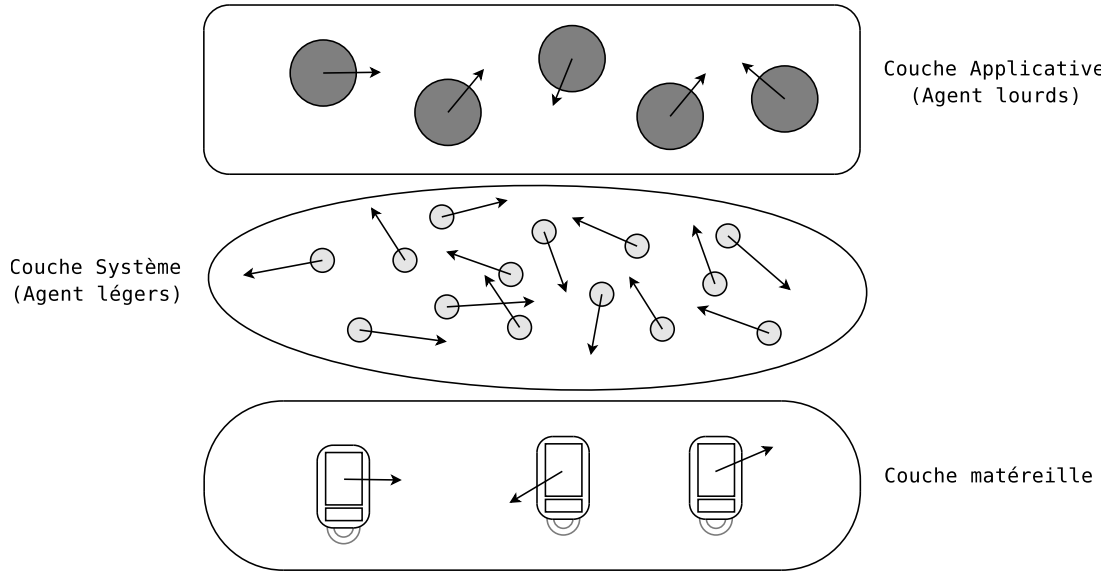


FIG. 3.8 – Les couches d'agents

Pour pouvoir faire ces choix optimisant leurs migrations, ils ont besoin d'avoir une représentation, logicielle et matérielle, qui leur garantissent une probabilité élevée de trouver à l'endroit indiqué les composants recherchés.

Cette représentation va être offerte par la couche d'ambiance qui sera une sorte d'aura à laquelle vont se référencer les agents lourds pour être aiguillés sur leurs choix de migration (fig 3.8), aussi bien sur la localisation de leurs partenaires que sur les itinéraires à suivre. Nous employons volontairement les termes *d'ambiance* et *d'aura* car ils représentent le temps certain que va mettre cette couche à absorber les modifications physiques avant de mettre à jour sa représentation. C'est pour cette raison qu'elle ne garantit pas avec certitude les informations qu'elle délivre mais qu'elle offre une certaine probabilité de réussite aux agents lourds lorsqu'ils devront faire un choix de déplacement. Cependant, la réactivité de la couche d'ambiance doit être la plus grande possible et va dépendre principalement du rapport entre les mobilités des agents légers et des sites supports. Nous étudions cet aspect dans le chapitre 6.

3.5 Conclusion

Dans la partie sur l'état de l'art, nous avons vu que les architectures hybrides cherchent à offrir l'ubiquité de la connexion à Internet aux utilisateurs nomades. Dans ce chapitre, nous avons étudié les problèmes que vont poser des tels environnements dynamiques lorsqu'ils sont déployés sur une large échelle. Pour commencer, la **mobilité matérielle**, induite par les unités mobiles, ne peut plus être gérée de manière hiérarchique et centralisée comme le proposent les technologies d'interconnexion de réseau. On ne peut plus disposer d'un mécanisme fiable et rapide permettant d'identifier et d'atteindre n'importe quel élément d'un internet. Sans ce mécanisme, on ne pourra plus mettre en place des communications à distance fiables et permanentes. La mobilité physique impose donc que le système prenne uniquement en charge le contexte local et que la migration s'effectue avec les sites voisins.

D'un autre côté, la mobilité matérielle justifie le choix d'utiliser les agents mobiles car ils permettent des interactions purement locales, sans risque d'interruption due au réseau. Ils sont autonomes vis à vis de tout système central et adaptables aux changements de l'environnement. Mais l'utilisation des agents va induire un deuxième niveau de dynamisme dû à la **mobilité logicielle**. Nous sommes donc confrontés à une **double mobilité**, logicielle et matérielle, qui ne pourra pas être prise en charge par le système qui se limite à une gestion du contexte local.

Avec cette double mobilité et avec une gestion locale, le concepteur devra lui-même déterminer la *localisation* des services qu'il souhaite utiliser et les *chemins* pour les atteindre. Les services étant réalisés par des agents mobiles, la localisation consiste à déterminer sur quel site se trouve le service désiré. Une fois la localisation obtenue, le concepteur devra déterminer le chemin permettant à son agent d'atteindre le service visé. La migration étant limitée aux voisins immédiats d'un site, un chemin est donc défini comme une succession de migrations de proche en proche.

Le second problème de ces architecture hybrides, c'est qu'elles s'inscrivent dans une large échelle. En effet, le nombre croissant d'utilisateurs nomades, à travers la téléphonie mobile par exemple, et l'augmentation du nombre de services accessibles sur Internet (ex. WebServices) accroissent considérablement l'échelle de déploiement d'une application. Cette augmentation de l'échelle pose un problème au niveau de la gestion de la mobilité. Le nombre important de sites augmente la difficulté de détermination de la localisation et une échelle planétaire augmente la taille des chemins. De plus, le dynamisme des environnements pourra remettre en cause la localisation et le chemin entre chaque migration.

Afin de proposer une solution pour gérer la double mobilité, nous avons donné différents critères permettant de classifier les agents mobiles présents dans l'environnement. Nous avons identifié deux grands types d'agents mobiles : les **agents lourds** et les **agents légers**. Les agents lourds vont représenter les applications métiers qui possèdent un savoir faire complexe et cherchent à utiliser des services précis. Ils vont se déplacer lentement, faire de longues visites et utiliser la migration ciblée. Ce sont ces agents qui sont confrontés aux problèmes de la double mobilité. Les agents légers, quant à eux, s'occupent de petites tâches et sont totalement indépendants, *i.e* ils ne cherchent pas de services précis. Ils se déplacent rapidement, font de courtes visites et effectuent des migrations libres. Les agents légers ne subissent pas le problème de la double mobilité.

A partir de ces deux types d'agents, nous pensons que les agents légers peuvent servir pour décrire des services systèmes utilisables par les agents lourds. Nous proposons donc de réaliser une **couche d'ambiance** qui va prendre en charge une partie de la gestion de la répartition. Cette couche est réalisée par les agents légers qui peuvent se rendre compte des changements de l'environnement grâce à leurs déplacements rapides et fréquents. Le but de la couche d'ambiance est d'offrir un environnements plus stable aux agents lourds. Mais elle ne pourra pas garantir l'exactitude de la représentation de l'environnement car il lui faudra un certain temps pour absorber les modifications du contexte.

Dans le chapitre suivant, nous étudions plus précisément une méthode permettant à la couche d'ambiance de prendre en compte la gestion de la répartition dans des environnements doublement mobiles. Pour cela, nous étudions plus précisément les méthodes de gestion de la répartition dans des environnements dynamiques, puis nous déterminons les besoins spécifiques des applications métiers et, enfin, nous mettons en évidence l'adéquation de l'utilisation des agents légers pour construire la couche d'ambiance.

Gestion de la répartition et adéquation des agents légers

4

Dans le chapitre précédent, nous avons identifié la double mobilité comme le principal problème des architectures hybrides à grande échelle, en l'occurrence le système ne peut plus prendre en charge la gestion de la répartition. À partir de cette constatation, nous devons regarder quelles en sont les implications pour la conception des applications réparties. Pour cela, nous étudions la gestion de la répartition dans les réseaux pair à pair. Nous verrons que cette absence de gestion de la répartition pose un problème pour la classe d'applications dites métiers.

Pour pallier à la gestion purement locale du système, nous proposons de mettre en place une représentation de l'environnement par une couche d'ambiance introduite précédemment. Celle-ci étant construite avec les agents légers, nous montrons qu'ils sont bien adaptés pour référencer les éléments présents dans l'environnement et constater les changements dus à la double mobilité. Pour terminer, nous expliquons le principe de fonctionnement de la couche d'ambiance.

4.1 La visibilité de la répartition

Lorsque l'on construit des applications distribuées, la gestion de la répartition est un point primordial. Soit celle-ci est explicite, *i.e* c'est au concepteur de mettre en place ses propres mécanismes de gestion, soit elle est gérée par le système qui offre une représentation globale aux concepteurs en lui assurant une localisation précise des services (pages jaunes) et une communication à distance fiable (ex. invocation de méthode à distance). Dans le cadre classique de conception, on essaye le plus possible d'utiliser la deuxième solution, *i.e* la gestion système, afin que le développeur dispose de l'ensemble des services comme s'ils étaient locaux et qu'il n'ait plus qu'à définir les différentes interactions nécessaires à la réalisation de sa tâche. Pour illustrer cette gestion système, nous pouvons citer à titre d'exemple l'intergiciel CORBA [GGM99]. Cette gestion du système est rendue possible grâce au réseau sous-jacent et à des services majoritairement statiques. Dans de telles conditions, c'est le modèle classique du client/serveur qui s'est imposé au fil des années.

Nous devons nous poser la question de savoir si cette méthode de conception peut rester valable à partir du moment où nous introduisons la double mobilité et la large échelle présentées dans le chapitre précédent. Pour y répondre, nous commençons par regarder quelles différences, dans la gestion de la répartition, sont apparues avec la mise en place des réseaux logiciels *pair à pair* [Dun01]. Dans ces réseaux, où chaque élément est à la fois client et serveur,

n'importe quel participant peut apparaître et/ou disparaître à tout instant. Ceci représente un des premiers contextes fortement dynamique à grande échelle. Dans cette architecture, la principale activité est de partager et/ou de rechercher des fichiers qui sont fragmentés et répartis sur un ensemble de pairs. Le problème majeur consiste à trouver l'ensemble de ces fragments afin de récupérer localement un copie complète du fichier recherché.

Pour résoudre ce problème de recherche, le premier mécanisme était de disposer d'un service central de publication (Napster) où chaque participant venait proposer les ressources qu'il acceptait de partager. Cette solution fut rapidement abandonnée pour plusieurs raisons dont, principalement, sa gestion centralisée qui était fragilisée dans un contexte fortement dynamique car une défaillance du service central rendrait inutilisable l'architecture. De plus, elle ne respecte pas la philosophie des réseaux P2P où tous les éléments sont égaux et gèrent eux-mêmes la répartition. La deuxième solution va donc abolir complètement la structure centralisée en se basant uniquement sur les éléments pairs. Pour cela, elle utilise des mécanismes classiques d'inondation entre voisins afin de trouver et de partager leurs ressources (Gnutella). Cette méthode fut elle aussi rapidement abandonnée car les apparitions et disparitions incessantes des sites provoquaient une importante surcharge du réseau due aux inondations nécessaires à la prise en compte des modifications.

En réalité, le principal problème de ces architectures dynamiques est de disposer d'une représentation générale de l'environnement qui permette aux nouveaux arrivants de connaître rapidement l'emplacement des éléments recherchés. Pour offrir cette représentation, la solution la plus couramment utilisée à l'heure actuelle est de mettre en place un ensemble d'éléments relativement stables qui permet de proposer un noyau fixe auquel se rattachent les nouveaux arrivants. Ces derniers s'adressent directement au noyau pour connaître la localisation des ressources recherchées.

Nous voyons ici que l'introduction d'un premier niveau de dynamisme remet en cause la gestion de la répartition par un serveur centralisé, alors qu'elle est la principale référence du modèle client/serveur classique. De plus, il n'est pas non plus envisageable de laisser la gestion intégrale de l'environnement aux sites. Dans ces conditions, il est donc difficile d'obtenir une représentation globale de l'environnement, bien que le niveau de dynamisme reste relativement restreint. En effet, dans les architectures P2P, un élément reste assez longtemps dans l'environnement pour permettre l'établissement ininterrompu de connexions à distance durant la récupération d'un fragment convoité. Ceci ne sera pas le cas dans les environnements fortement dynamiques comme les architectures hybrides.

Si nous voulons à présent passer dans des environnements complètement dynamiques, nous devons considérer alors que les sites peuvent apparaître, disparaître et se déplacer, que les ressources deviennent mobiles et que les éléments recherchés ne sont plus uniquement des morceaux de fichiers à télécharger mais des services complets. Dans ce cas, nous ne pouvons plus considérer que les éléments « *pairs* » sont les sites mais directement les agents. Les sites n'étant alors qu'un support pour les agents, ils n'offrent qu'une gestion de leur contexte local.

4.2 Le problème des applications métiers

Nous venons de voir que le système ne peut pas prendre en charge une représentation globale de l'environnement. La gestion de la répartition revient donc au concepteur. Cette absence de représentation globale pose un problème dans le cadre des *applications métiers*. Une application métier est définie par un savoir complexe s'appliquant dans un domaine précis.

Une application de fouille de données (datamining), qui permet de dénicher des tendances ou des corrélations cachées parmi des masses de données, est un exemple d'application métier. Lorsque l'application est répartie, elle connaît parfaitement les services à utiliser et l'ordre dans lesquels ils doivent être sollicités. Les applications métiers vont donc se baser sur des agents volumineux, à cause du savoir faire complexe qu'il transporte, et se déplaçant en fonction du service à mettre en œuvre. Ces deux propriétés correspondent à la description des agents lourds utilisant la migration ciblée.

Le problème de la migration ciblée est qu'en l'absence d'une représentation globale du contexte, ce sont les agents eux-mêmes qui devront parcourir l'environnement pour trouver les partenaires cherchés. Or, pour les applications métiers, c'est la phase de traitement local et les coopérations qui sont importantes, les déplacements sont eux secondaires, voir ils représentent une perte de temps. Mais les agents lourds possèdent deux principales caractéristiques qui ne leur permettront pas d'avoir des déplacements efficaces et donc de s'adapter facilement au contexte dynamique.

Premièrement, avec leur volume important, les agents lourds vont avoir une vitesse de déplacement réduite. Cette lenteur ne leur permettra pas d'atteindre rapidement leur cible et pourra induire des déplacements inutiles. Par exemple, si un agent lourd utilise un chemin trop long pour rejoindre sa cible, le service visé, ou le site support, peut avoir changé de localisation. Deuxièmement, le savoir faire particulier à chaque agent étant relativement complexe, il va imposer de longues phases de calcul local et de longues phases de coopération. Dans un environnement dynamique, lorsque les visites sont longues, un agent peut trouver un contexte complètement différent entre le début et la fin de sa visite, cela nécessite de prendre en compte un important nombre de changements à chaque fin de visite. Nous devons donc proposer une méthode permettant de minimiser le nombre de déplacements nécessaires à un agent lourd pour atteindre les partenaires visés sans être affecté par les changements de l'environnement.

Ensuite, d'un point de vue de la conception dans le cadre des applications métiers, nous devons soulager le concepteur d'un certain nombre de contraintes liées au système afin qu'il puisse concentrer son effort de conception sur la description des tâches de chaque agent composant son application. En effet, la complexité des applications métiers ne doit pas être augmentée par la difficulté de prise en compte de la double mobilité. Généralement, les concepteurs cherchent à s'abstraire le plus possible des problèmes liés à l'environnement afin de raisonner directement au niveau applicatif en se reposant sur des services système fiables.

Pour faciliter la tâche de conception des applications métiers, nous devons donc proposer un service système qui permette de connaître l'état courant de l'environnement afin de pouvoir aiguiller efficacement les agents lourds. Nous avons vu que les sites ne pourront pas prendre à leur compte cette gestion du contexte général. Nous devons donc la confier aux agents. Nous choisissons alors de réaliser un service à base d'agents qui s'occupera de la représentation du contexte et qui sera directement utilisable par les agents lourds à travers de simples phases de coopération. Les agents lourds travailleront directement avec d'autres agents, entre pair, et pourront récupérer directement les informations nécessaires à l'optimisation de leurs déplacements.

4.3 Une gestion de la répartition à base d'agents légers

Nous avons vu que la brique de base assurée normalement par le système, qui est la représentation du contexte global associée à une communication distante, n'est plus envisageable

dès lors que l'architecture réseau devient dynamique et les services mobiles (cf 3.1). Plusieurs études ont proposé d'utiliser les agents (proches du domaine des SMA) afin de capter l'évolution des environnements fortement dynamiques [RCBL04] grâce à leurs capacité d'autonomie, de réactivité et d'adaptabilité. Notre approche s'inscrit dans cette démarche, mais nous nous sommes ici focalisés sur l'apport de la mobilité et limité à notre modèle d'agent inscrit dans le domaine des systèmes distribués. Nous proposons donc une méthode alternative pour la gestion de la répartition qui devra s'appliquer aux environnements où le système offre uniquement une gestion locale pour la communication et pour la représentation du contexte. Pour cela nous devons identifier les contraintes propres de ce type d'environnement en regardant comment les intégrer.

La première contrainte vient de l'accessibilité discontinue des sites due à leurs déplacements et imposant l'architecture dynamique du réseau. Nous avons vu que ces sites doivent être considérés uniquement comme des supports d'exécution et que l'entité principale de l'environnement sera l'agent. De plus cette mobilité physique induisant une faible fiabilité des communications à distance, les agents devront communiquer localement. Il faut donc concevoir la gestion de la répartition par des agents qui sont complètement découplés des sites, qui pourront évoluer sans avoir besoin d'une représentation globale de l'environnement physique et qui se contentent d'établir des communications locales.

La deuxième contrainte vient de la mobilité des services et de leur construction sur un ensemble d'agents. Auparavant, le concepteur pouvait connaître directement le service à utiliser ainsi que la manière de le contacter. Par exemple, dans le cas d'un service ftp, le nom du serveur détermine directement sa localisation. Cela n'est plus possible avec la double mobilité où on ne peut avoir une localisation fixe pour un service donné. Une autre solution consiste à utiliser un service de nommage, comme dans CORBA, qui permet de récupérer dynamiquement la localisation d'un service précis. Mais la multitude des agents pouvant composer une application, couplée à leurs déplacements, nécessiterait des mises à jour trop fréquentes pour que cette solution soit encore possible. La gestion de la répartition devra donc utiliser une méthode complètement décentralisée où chaque agent peut évoluer sans avoir à référencer un service de nommage centralisé. Au fur et à mesure de leurs déplacements, les agents pourront trouver, sur leur chemin, les ressources réparties à intégrer dans la représentation globale.

La solution que nous proposons pour prendre en compte ces deux contraintes est de définir la représentation de la répartition sur des agents respectant certains des critères que nous avons définis (cf 3.3.1). Pour se détacher des problèmes de localisation, les agents devront utiliser la migration libre et identifier leurs partenaires au hasard. Pour s'affranchir du problème des déplacements des sites, ils devront être très mobiles en minimisant le temps des coopérations locales et en limitant leur volume. Cet ensemble de caractéristiques correspond parfaitement aux agents légers qui nous avons définis dans la partie 3.3.1.

Pour prendre en compte les contraintes liées à la double mobilité, nos proposons donc de concevoir la représentation du contexte à partir d'un ensemble d'agents légers. Ces agents devront être parfaitement indépendants les uns des autres et ne chercherons pas à atteindre une destination préétablie. La difficulté principale de conception résidera alors dans la bonne description des tâches individuelles de chaque agent et de la bonne définition de leurs brèves coopérations locales. Par contre, le nombre des agents n'est pas limité et pourra être assez grand pour réaliser l'ensemble des tâches nécessaires.

4.4 Le service offert par les agents légers

Nous présentons maintenant plus en détail le service proposé aux agents lourds pour les aiguiller dans leurs déplacements afin d'optimiser la recherche de partenaires. Ce service va offrir une représentation de l'environnement global permettant aux agents lourds de localiser plus simplement leurs partenaires et de déterminer plus facilement le chemin pour les rejoindre. Mais avant de détailler ce service nous devons voir si les agents légers sont bien adaptés pour accomplir la tâche que nous attendons d'eux.

4.4.1 Quel type d'agent utiliser

Pour cela, rappelons que les sites gèrent uniquement leur contexte local, *i.e* ils connaissent uniquement les agents en cours d'exécution localement et les autres noeuds accessibles dans le voisinage (cf 3.1.1). Pour pouvoir proposer une représentation plus globale, nous devons utiliser des agents qui soient capables d'explorer rapidement le reste de l'environnement afin de connaître les éléments distants, puis de fournir aux agents lourds les informations récoltées. Celles-ci doit être délivrées avant que l'environnement n'ait à nouveau changé. La capacité de déplacements rapides des agents légers répond à cette contrainte.

Cependant, nous pouvons facilement envisager que cette technique pourra fonctionner pour le proche environnement d'un site, mais ne pourra pas fonctionner si l'agent doit parcourir lui-même l'intégralité des noeuds présents dans le réseau. En utilisant tout un ensemble d'agents se déplaçant sur le réseau, nous pouvons répartir le nombre de sites à visiter et donc diminuer le nombre de visites à faire par chaque agent. Ainsi, l'exploration générale sera effectuée plus rapidement, grâce à plus de parallélisme, et en délivrant localement l'ensemble des informations recueillies, nous obtenons la représentation générale souhaitée. Mais, pour cela, nous allons nous heurter à trois principaux problèmes, à savoir la taille des données véhiculées, la pertinence des informations récoltées et la manière dont les agents vont délivrer ce qu'ils ont découvert.

D'abord, lors de ses déplacements, un agent va accumuler de plus en plus d'informations sur les éléments qu'il va découvrir en augmentant ainsi son propre volume au fur et à mesure de ses explorations. Donc si nous laissons un agent accumuler trop d'information, il va perdre sa rapidité de déplacement nécessaire à une exploration efficace de l'environnement. Nous devons donc permettre aux agents de garder une taille critique en leur offrant la possibilité de se séparer d'une partie des informations qu'ils ont déjà accumulées en la laissant, par exemple, sur les sites de visite.

Ensuite, l'agent ne devra pas garder trop longtemps une information au risque de la rendre obsolète. Ceci est particulièrement vrai avec la double mobilité où les sites et les agents vont régulièrement se déplacer. Chaque agent devra donc être capable d'évaluer la pertinence des informations qu'il transporte. Ceci est assez facile pour les agents mobiles qui n'auront pas besoin d'une horloge mais simplement d'exprimer la datation en fonction du nombre de migrations. Ainsi plus le nombre de migrations après la récupération d'une information est grand et plus la pertinence de l'information décroît. A partir de cette datation, l'agent pourra alors délivrer les informations qu'il estime valables et se débarrasser des autres.

Enfin, nous avons vu que pour des raisons évidentes de tolérance aux fautes dans les environnements dynamiques, les agents doivent être le plus indépendant possible des noeuds. Par conséquent, nous ne pourrons pas imposer aux agents de retourner sur un site prédéfini afin

de lui donner les informations collectées. De plus, pour assurer la continuité du service malgré la disparition d'un ou plusieurs agents, ces derniers devront être totalement indépendants les uns des autres. Ceci impose que ce sont les agents individuellement qui vont choisir les parties du réseau à explorer et les sites où ils déposent les informations récoltées.

Nous venons donc de voir que la réalisation du service qui offre une représentation globale de l'environnement nécessite des caractéristiques précises pour les agents. Il s'agit d'une capacité de déplacement rapide, d'une forte indépendance et d'un nombre important d'agents. Cet ensemble de propriétés correspond parfaitement aux agents légers qui vont donc servir de base pour la réalisation du service proposé aux agents lourds.

4.4.2 Principe de fonctionnement

Nous avons vu que les deux principaux problèmes liés à l'utilisation de la migration ciblée, utilisée par les applications métiers, sont la localisation d'un partenaire et la détermination du chemin pour le rejoindre. La méthode que nous proposons ici va permettre de prendre en compte ces deux cas, méthode dont la conception va s'avérer simplifiée par l'utilisation des agents légers.

Le principe de fonctionnement va se scinder en deux parties, une partie exploratoire et une partie de reconstruction locale. La partie exploratoire est assurée par un ensemble d'agents légers qui fonctionnent tous selon un même comportement cyclique. L'agent arrive sur un site, il dépose les informations qu'il transporte, puis il récupère auprès du système les différents agents présents sur le site ainsi que les sites voisins accessibles. Ensuite, il marque ces nouvelles données comme issues de la visite en cours, après quoi il supprime les informations les plus anciennes, en fonction du nombre de migrations effectuées, et enfin il migre vers un autre site où il recommence ce comportement.

La partie reconstruction est assurée par un agent local auquel s'adresse l'agent léger lorsqu'il dépose ses informations. En fait, l'agent local va posséder sa propre représentation de l'environnement global qu'il va mettre à jour en fonction des données que vont lui laisser les agents en visite. Chaque fois qu'un agent dépose des informations, l'agent local les compare avec celles qu'il possède déjà et effectue ensuite les éventuelles modifications nécessaires pour mettre à jour sa représentation globale.

Nous voyons donc que le service recherché peut se construire simplement grâce à l'utilisation des agents légers où aucune interaction entre les agents en déplacement n'est nécessaire. De plus, ils effectuent une tâche peu complexe et les phases de coopération sont brèves. Ces éléments garantissent ainsi leur indépendance, leur rapidité de déplacement et par conséquent leur adaptation à la double mobilité. Les agents lourds pourront donc simplement s'adresser à l'agent local afin de récupérer la représentation de l'environnement global et ainsi faire des choix d'optimisation de leurs déplacements par rapport aux partenaires qu'ils souhaitent atteindre.

4.5 Conclusion

À partir de l'étude des réseaux pair à pair qui constituent les premiers réseaux dynamiques à grande échelle, nous avons pu mettre en avant que le système ne pourra pas assurer la gestion de la répartition. Ainsi, la distribution des éléments, logiciels et matériels, au sein de l'environnement est explicite et doit être gérée directement par le concepteur des applications réparties.

Cependant, les applications métiers vont avoir du mal à prendre en compte directement le dynamisme de l'environnement. En effet, ces applications sont construites sur des agents qui sont lents, font de longues visites et utilisent la migration ciblée. Ces caractéristiques ne sont pas compatibles avec la double mobilité, car elles ne permettent pas aux agents de s'adapter facilement à leur environnement. De plus, dans le cadre des applications métiers, l'effort de conception doit être fait sur le savoir propre de l'application et doit être soulagé des problèmes systèmes.

Pour ces deux raisons, nous avons proposé de mettre en place une gestion de la répartition basée sur les agents légers. Grâce à leur rapidité de déplacement et leur autonomie, nous avons expliqué que ce type d'agents est parfaitement adapté pour référencer tout type d'éléments et en constater les changements. Nous avons donc exposé le service rendu par les agents légers ainsi que son principe de fonctionnement. Rappelons que la couche d'ambiance ne garantit pas la répartition des éléments mais a pour but de proposer un environnement plus stable pour les agents lourds qui doivent toujours gérer un partie de la distribution.

Le modèle des agents mobiles coopérants

5

Dans les deux précédents chapitres, nous avons introduit le phénomène de la double mobilité, physique et logicielle, que nous souhaitons maîtriser grâce à l'utilisation de deux grands types d'agents : les agents légers et les agents lourds. Cette distinction nous permet d'envisager la réalisation grâce aux agents légers d'une couche dite « *d'ambiance* » offrant aux agents lourds une représentation plus stable et plus cohérente de l'environnement.

Dans ce chapitre nous nous attelons à la description plus précise du modèle des agents mobiles coopérants à travers les différents éléments que nous devons mettre en œuvre pour permettre la réalisation d'une application à base d'agents mobiles. Ces éléments doivent suffire aux agents légers pour qu'ils puissent construire la couche d'ambiance désirée.

Nous allons commencer par une brève présentation, sous forme de rappels, des différents choix que nous avons fait pour la conception des applications, puis nous exposons précisément l'ensemble des éléments nécessaires pour la définition et la gestion d'un agent. Ensuite nous nous attaquons aux services de base offerts par le système et enfin nous terminons par la présentation d'un protocole de visite permettant de réaliser les coopérations présentées dans la section 2.2.

5.1 Présentation générale

Dans le chapitre 2 nous avons fait un certain nombre de choix portant aussi bien sur les éléments de base, comme la communication ou la méthode de migration, que sur l'environnement de développement. Nous avons aussi précisé que la méthode applicative de communication entre les agents se base sur la mise en place de phases de coopération. Le modèle d'agents mobiles que nous présentons ici doit respecter l'ensemble de ces critères et doit offrir les éléments minimaux pour simplifier la conception d'applications réparties sur les environnements dynamiques et en particulier celle de la couche d'ambiance grâce aux agents légers. Tout ces critères, pris en compte par le modèle, sont rappelés dans cette présentation générale.

Pour commencer, nous avons choisi de proposer un système complètement basé sur la notion d'agents et utilisant les sites uniquement comme support d'exécution. Ceci veut dire que quels que soient les services de base réalisés par le système (cf 5.3), ils devront être directement utilisables par les agents et proposer des mécanismes simples de mise en route, en particulier pour l'exécution, la migration et la coopération.

Ensuite, nous avons pris la décision de définir un système ne permettant pas la communication distante, celle-ci étant exclusivement utilisé pour la migration des agents, et privilégiant les interactions locales par l'utilisation de la coopération. Pour minimiser cette restriction, la mise en relation des agents locaux doit être facilitée par un annuaire permettant de référencer tous les agents souhaitant établir des coopérations locales. Cet annuaire permettra en plus de proposer les règles de coordination propres à chaque agent. N'oublions pas que chaque agent possède sa propre politique générale de coopération qu'il pourra aussi divulguer, s'il le souhaite, aux autres agents grâce à ce service d'annuaire.

Puis nous avons vu aussi que nous souhaitons proposer un modèle se basant sur les notions de comportements/transitions (cf 1.3.2) permettant de maîtriser précisément les différentes étapes des agents et principalement celles intervenant durant les phases d'exécution locale. Notre modèle va donc fournir un comportement standard de visite ainsi qu'un comportement standard de coordination (cf 5.4) que devra respecter chaque agent afin de garantir le déroulement sans interruption des coopérations et de garder le contrôle de leur propre évolution. Nous nommons ces comportements standards « *protocoles* » afin de différencier les comportements imposés par le système de ceux créés par le concepteur. Ces protocoles servent à faciliter la conception aux développeurs qui pourront les utiliser tels quels ou les enrichir à leur guise.

Enfin pour terminer, nous rappelons que nous optons pour des agents proactifs et, par conséquence, d'un point de vue système, ce sont les agents qui assument leurs déplacements. Pour leur permettre de choisir au mieux leurs destinations, les agents doivent connaître facilement les différents emplacements atteignables. Pour cela, nous avons besoin d'identifier précisément chaque site à l'aide d'un identifiant unique et de trouver, auprès d'un service de base, l'ensemble des éléments appartenant directement au voisinage du site. Pour cela nous définissons l'identification d'un site comme suit.

Définition 5.1 (Localisation de site) *chaque site possède un couple formé de son nom et du numéro de série unique attaché au périphérique de communication. Si un même site possède plusieurs périphériques de communication, il sera considéré comme deux éléments distincts au niveau du système.*

Avec ces rappels effectués, nous avons clairement identifié les besoins auxquels doit répondre notre modèle d'agents mobiles coopérants que nous pouvons à présent regarder plus en détail en commençant par une définition précise des agents.

5.2 Définition d'un agent

Dans le premier chapitre (cf 1.3.1) nous définissions un agent comme un élément autonome possédant une activité interne avec ses propres ressources, œuvrant généralement au nom d'un utilisateur ou d'une application, communiquant avec d'autres agents afin de réaliser la tâche pour laquelle il a été créé [MDW99].

Nous précisions aussi que, dans les applications réparties, ce modèle s'étend en donnant la possibilité aux agents de se déplacer de sites en sites. Cette mobilité peut être contrôlée par le système, on parle alors d'agents mobiles réactifs, ou par les agents eux-mêmes, on parle alors d'agents mobiles proactifs [FPV98]. Nous choisissons d'utiliser des agents proactifs.

Nous gardons cette définition pour notre modèle où nos agents mobiles sont des entités mobiles, autonomes et proactives. Maintenant, pour permettre leur intégration dans l'intergiciel, nous avons besoin d'identifier précisément chaque agent.

5.2.1 L'identification

Pour que le système puisse contrôler les agents et les enregistrer dans un service d'annuaire, nous avons besoin d'identifier, sans ambiguïté, chaque agent du système. Cette identification doit être partageable et gérée de la même manière sur l'ensemble du système car elle servira lors des migrations ciblées qui peuvent référencer n'importe quel agent dans tout l'environnement. Nous proposons donc la définition suivante de l'identification des agents :

Définition 5.2 (Identification d'un agent) *Un agent est défini par un identifiant unique non-réutilisable engendré lors de sa création et qu'il conservera jusqu'au terme de son cycle de vie. On utilise par exemple la date et le site de création de l'agent.*

5.2.2 Les propriétés

La définition que nous venons de donner suffit pour désigner un agent dans le système, gérer son exécution et réaliser ses migrations. Cependant, elle ne suffit pas pour enrichir l'annuaire et ainsi permettre de faire des recherches de partenaires basées sur l'application créatrice ou encore les services proposés, recherche qui vont s'avérer nécessaires pour la mise en place de certaines coopérations.

De plus, nous avons besoin de publier les protocoles de coopération propres à chaque service que souhaite proposer un agent. Pour ce faire, nous utilisons un service d'annuaire que nous présenterons plus tard et qui permettra de référencer à partir d'un identifiant toutes les propriétés d'un agent allant de son nom jusqu'aux services proposés. Nous définissons cet ensemble de propriétés accessibles comme suit :

Définition 5.3 (Propriétés d'un agent) *Un agent possède un ensemble de propriétés accessibles à partir d'une interface minimale commune à tous les agents.*

Cet ensemble de propriétés comportera un minimum d'informations qui permettra d'identifier les origines de l'agent, ainsi que les différents services rendus accompagnés des protocoles de coordination permettant de les utiliser. Nous obtenons la liste suivante pour les propriétés d'un agent :

- l'identifiant
- le site de création
- le propriétaire,
- l'application génitrice,
- l'ensemble des services proposés accompagnés de chaque protocole de coordination correspondant.

Cette liste est accessible grâce à l'interface suivante :

```
Liste unAgent.Propriétés()
```

Avec ces deux premières définitions, nous avons les éléments minimaux permettant de réaliser des applications utilisant la migration libre. En effet, l'identification permet au système d'effectuer les différentes migrations souhaitées et les agents pourront réaliser les rencontres

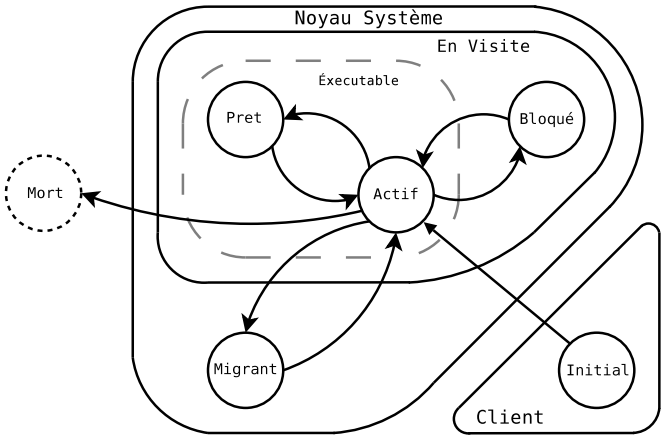


FIG. 5.1 – Automate d'états d'un agent

involontaires en récupérant, auprès du service d'annuaire, l'ensemble des partenaires locaux souhaitant coopérer. Par contre, dans le cas de la migration ciblée nous avons besoin de déterminer précisément l'emplacement des agents visés.

5.2.3 La localisation

Nous avons vu que la migration ciblée peut s'avérer très compliquée à engager à cause de la double mobilité. Pour éviter les déplacements inutiles, il est nécessaire de pouvoir déterminer sans ambiguïté la localisation d'un agent. Pour cela nous allons utiliser l'identifiant unique de chaque site spécifié dans la définition 5.1

Définition 5.4 (Localisation d'un agent) *La localisation d'un agent correspond à l'identifiant du site sur lequel il s'exécute (momentanément).*

Nous sommes donc en mesure de repérer un agent grâce à son identifiant et sa localisation qui sont les deux éléments nécessaires et suffisants pour réaliser des applications basées sur la migration ciblée. Les agents pourront déterminer la localisation du service qu'ils souhaitent utiliser, migrer vers l'emplacement identifié en fonction d'un chemin déterminé et enfin mettre en route la coopération souhaitée qui permettra de réaliser le service désiré. Rappelons que la localisation et le chemin doivent être pris en charge par le concepteur de l'agent.

5.2.4 Automate principal

Comme dans tout système multi-tâches, la présence de plusieurs agents au même instant sur un site va nécessiter des mécanismes permettant de partager les ressources systèmes en fonction des demandes des agents. Pour cela, nous nous inspirons des modèles d'automates à états pour les processus que nous trouvons dans la gestion des systèmes d'exploitation. Les deux principales différences vont venir de la manière de créer les agents et surtout de la représentation de la migration. Nous représentons les différents états accessibles par un agent de notre modèle par l'automate de la figure 5.1.

Nous avons fait plusieurs distinctions sur le schéma afin de mettre en avant les différentes possibilités d'évolution d'un agent. La première concerne les états dit «exécutables»

qui permettent de distinguer les transitions déclenchées par le système de celles provoquées par l'agent lui-même. Les modifications d'état exécutable, imposés par le système, permettent de garantir un ordonnancement équitable des différents agents en cours d'exécution, l'égalité pouvant être établie selon différents critères propre à l'intergiciel, comme le quantum de temps par exemple. L'agent n'a pas de contrôle sur le passage de l'état actif à l'état prêt au contraire de toutes les autres transitions qu'il exécute lui même.

La seconde distinction porte sur les états que peut atteindre l'agent lorsqu'il est en cours de visite sur un site. C'est l'agent lui-même qui décide de passer de l'état actif à l'état bloqué, principalement dans le cadre des coopérations, ou à l'état de migration avant un déplacement. Ces transitions ne sont pas dues aux décisions du système mais bien à l'agent lui-même, garantissant ainsi son caractère proactif. Nous avons séparé la migration car il s'agit d'un état particulier du système dans le sens où une fois le déplacement engagé par l'agent, c'est le système qui le remettra dans l'état actif et qu'un changement de localisation impose de le retirer de l'environnement d'exécution de départ pour l'intégrer dans celui d'arrivé.

La toute dernière distinction porte sur la création des agents qui est extérieure au noyau du système. Comme les sites ne servent que de support d'exécution, le système n'a pas la possibilité de créer directement des agents et servira uniquement de service de démarrage aux agents venant des applications. Ce sont donc les applications qui créent les agents à mettre en route et les proposent au système pour qu'il les prenne en charge et lance leur exécution.

Pour permettre de bien mettre en place ces distinctions d'états, nous proposons plusieurs interfaces qui seront utilisées par le système pour effectuer les transitions. La première est `Arriver()` qui est appelée par le système juste après la création de l'agent et qui place l'agent dans l'état exécutable. La seconde est `Visiter()` qui permet de démarrer la tâche applicative de l'agent et le place dans l'état visite. La dernière est `Terminer()` qui permet de sortir l'agent de l'état de visite avant une migration ou une terminaison complète.

```
unAgent.Arriver()

unAgent.Visiter()

unAgent.Terminer()
```

Ces trois interfaces correspondent à un ensemble d'actions qui sont définies par le concepteur lui permettant de concevoir un ensemble d'actions à réaliser durant les différentes phases rencontrées par ses agents. Par exemple, avec la fonction `Arriver()`, il pourra prendre connaissance du contexte afin de savoir quel comportement adopter. Nous verrons comment sont utilisées ces différentes fonctions dans la description des services de base.

5.2.5 Comportements et transitions

Nous avons vu dans la conclusion sur les intergiciels (cf 2.1.4), que le développeur doit maîtriser les différentes états atteints par ses agents au cours de leur exécution. Ceux-ci doivent être clairement définis lors de la conception. Pour cela, nous déterminons la tâche générale d'un agent à partir d'un ensemble de comportements/transitions comme décrit dans la définition suivante :

Définition 5.5 (Comportements - transitions) *La tâche générale d'un agent est définie à partir d'un ensemble de comportements, pouvant se réduire à un singleton, et un ensemble de transitions permettant de passer d'un comportement à un autre. Si l'agent souhaite migrer, deux comportements standard doivent être atteints, il s'agit de* ***Pré-migration*** *pour enclencher le déplacement et* ***Post-migration*** *pour la reprise de l'agent.*

Tous ces comportements sont atteignables dans l'état actif de l'automate général et pourront provoquer le passage à l'état bloqué ou à l'état de migration. En fait, le concepteur devra définir un automate applicatif, propre à chaque agent, lui permettant de définir précisément les différentes sous-tâches et particulièrement les comportements coopératifs qu'il souhaite engager.

Avec cette dernière définition pour les agents, nous venons de fixer les éléments essentiels permettant de définir et de gérer les agents au sein du système. Maintenant nous avons besoin de proposer un ensemble de services supports présents sur chaque site et permettant aux agents d'exécuter les actions élémentaires nécessaires à la réalisation de leurs tâches.

5.3 Les services de base du système

Nous présentons ici l'ensemble des services proposés par les sites et nécessaires à nos agents. Ces services sont déployés sur chaque site du système, accessibles uniquement localement et non bloquants. Nous définissons un service local comme suit :

Définition 5.6 (Service local) *Un service local est défini par une interface et implanté par un ensemble d'agents locaux, pouvant se réduire à un singleton, possédant un représentant référencé dans le service d'annuaire local.*

Chaque service fait donc partie intégrante du système, *i.e* il possède un représentant avec lequel les autres agents peuvent dialoguer directement. Les identifiants des représentants sont créés d'une manière identique, quel que soit le site, permettant ainsi aux visiteurs de retrouver facilement les services de base cherchés. De plus, tous les services sont référencés dans l'annuaire local du site qui sert ainsi de point de départ minimal pour connaître l'ensemble des autres services.

Pour faciliter la conception et l'écriture des algorithmes, les interfaces des services de base font partie intégrante de l'interface du site de visite. Ainsi, les agents accèdent directement aux service au travers d'une référence unique `ou` désignant le site en cours de visite. Maintenant, pour faire dialoguer, les applications avec les services de base, nous avons besoin de définir un mécanisme de communication précis et applicable à tout le système.

Définition 5.7 (Communication inter-agent) *La communication entre les agents s'effectue uniquement par l'appel de méthode locale. À chaque méthode d'un agent correspond une interface accessible grâce à l'identifiant de l'agent. Les interfaces de communications inter-agent sont publiques.*

Note : les interfaces correspondant à l'automate principal ne sont pas définies comme des interfaces de communication inter-agent mais comme des primitives système.

Le système propose tout un ensemble de mécanismes de diagnostic facultatifs permettant aux agents de savoir si les appels de méthode se sont déroulés correctement. La communication

s'effectue uniquement entre agents locaux à l'exception de ceux du service de migration qui auront besoin d'établir une communication distante pour transmettre les agents cherchant à se déplacer.

Avec ces deux définitions, nous pouvons maintenant identifier les services et établir les communications permettant aux agents de leurs adresser des requêtes. Nous présentons maintenant plus en détail chacun des services de base du système.

5.3.1 Le service d'exécution

Ce service permet aux agents de s'exécuter sur un site en réalisant l'automate principal proposé dans la section précédente. Lorsqu'un concepteur souhaite créer un agent, il adresse au service d'exécution un message contenant l'agent à activer grâce à l'interface suivante :

```
Verdict ou.Lancer(Agent unAgent)
```

Note : La référence *ou* désigne le site de visite courant.

Le développeur peut récupérer grâce au résultat de la fonction un verdict sur l'activation grâce à deux réponses standards. La première est ***Lancement réussi*** qui garantit la mise en route de l'agent et la seconde ***Lancement échoué*** informant de l'échec de la création.

Lorsque le lancement d'un agent est réussi, nous disons alors que l'agent exécute une visite sur le site, ce que nous avons mis en évidence sur l'automate d'état général de la figure 5.1. Le début de cette visite provoque l'activation d'un thread qui sera attaché à l'agent permettant de contrôler et d'exécuter l'activité de l'agent.

Nous découpons la visite d'un site en trois étapes : arriver, visiter et terminer. L'arrivée de l'agent peut correspondre à la création d'un nouvel agent ou à une migration, dans les deux cas elle provoquera la création d'un thread et l'appel à la fonction `Arriver()` de l'agent mis en route. La visite correspond au comportements applicatif de l'agent qui lui permet de réaliser ses traitements locaux et qui est activée grâce à l'interface `Visiter()` de l'agent. La terminaison correspond à la fin de la tâche de l'agent ou à sa migration. Dans tous les cas le thread qui lui était associé est détruit et sera précédé du lancement de la fonction `Terminer()` de l'agent.

Pour provoquer l'arrêt de l'agent, nous offrons deux possibilités. Soit l'agent arrive au terme de sa tâche et il termine lui même son exécution, soit c'est une intervention externe qui provoque la terminaison (l'application créatrice, le service de migration ou le système). Nous proposons donc l'interface pour la terminaison.

```
Verdict ou.Terminer(Agent a)
```

Note : l'agent pourra demander sa propre terminaison en utilisant la fonction avec le mot clé *this* le désignant : `ou.Terminer(this)`

Ce service est utilisé lors de la création d'un agent mais aussi lors d'une migration. Il permet aux agents de démarrer une exécution ou de la poursuivre suite à un déplacement. Le service de migration se sert de ces verdicts pour savoir si la migration a été intégralement réalisée.

5.3.2 Le service de voisinage

Ce service s'occupe de référencer les différents éléments atteignables dans le voisinage direct du site. Nous avons mentionné dans la partie 3.1.1 que la couche réseau était capable de récupérer les différents éléments présents dans le voisinage et d'obtenir leur identifiant afin de communiquer avec eux.

Le service de voisinage va s'occuper de maintenir à jour une base de données contenant l'ensemble des identifiants des sites présents dans le contexte local. Pour ce faire, elle interroge à intervalle régulier la couche réseau pour qu'elle sonde le contexte et lui envoie les éléments présents. Le service pourra alors mettre à jour sa base en ajoutant les nouveaux éléments et en retirant ceux qui ont disparu.

Nous avions vu aussi que les sites peuvent informer de leurs départs imminents en le signalant à la couche réseau. Lorsque cela arrive, la couche réseau récupère l'information et elle la fait remonter au service de voisinage qui fera les modifications nécessaires dans la base de voisins.

Pour récupérer les sites du voisinage, les agents peuvent adresser leurs requêtes grâce à trois interfaces permettant respectivement de trouver un site précis dans le voisinage, de choisir un voisin au hasard et de récupérer la liste entière de tous les voisins.

```
Site  ou.ExisteVoisin(Chaîne identifiant)

Site  ou.VoisinHasard()

Liste ou.Voisins()
```

Ces trois interfaces permettent aux agents de choisir eux-mêmes la future migration qu'ils souhaitent entreprendre. Si jamais la requête de l'agent ne peut être satisfaite, dans le cas d'un isolement du site par exemple, elles peuvent retourner *« null »*.

5.3.3 Le service de migration

Il permet aux agents de se déplacer vers un site donné. Lorsqu'un agent souhaite se déplacer il va se mettre volontairement dans l'état *Pré-migration* et va adresser un message au service de migration avec le nom du site qu'il souhaite atteindre. Ce changement d'état perdurera tant que le service de migration ne pourra pas statuer sur un verdict précis concernant la demande de l'agent. Pour réaliser une migration l'agent dispose de l'interface suivante :

```
Verdict ou.Migrer(Site destination)
```

Une fois averti, le service de migration contacte localement le service de voisinage des sites afin de savoir si la destination visée appartient bien au voisinage local. Si ce n'est pas le cas, alors le service de migration statue sur le verdict de ***Serveur inconnu***.

Après avoir eu confirmation de la présence du site désiré dans le voisinage, le service de migration origine fait une copie de l'agent, emballe le clone et l'envoie au service de migration destinataire grâce à un appel de procédure à distance. Il s'agit ici de la seule communication distante du système et nous utilisons l'appel de méthode à distance afin de garantir le déroulement intégral de la migration. Si l'établissement de l'appel de méthode ne peut se faire, le service de migration origine statue sur le verdict de ***Serveur non-joignable***.

```
Verdict dest.Accueillir(Agent migrant)
```

Note : La référence *dest* désigne le site de destination obtenue auprès du service de voisinage.

Si la connexion est établie, le service de migration origine attend de recevoir l'avis sur la reprise de l'exécution sur le site de destination. Pour cela, lorsque le service de migration destinataire reçoit l'agent en cours de migration, il le déballe et demande à son service d'exécution de lancer une nouvelle visite pour l'agent et attend son verdict. Si cette réponse est *Lancement échoué* alors le service de migration destinataire retourne le verdict de ***Serveur échec exécution*** au service de migration origine qui lui indique l'échec de la reprise de l'agent.

Par contre, si la réponse du service d'exécution destinataire est *Lancement réussi*, alors la migration s'est déroulée jusqu'à son terme sans rencontrer de difficulté. Le service de migration destinataire statue sur le verdict ***Migration effectuée*** qu'il retourne au service de migration origine lui indiquant ainsi le succès du déplacement. Le service de migration origine utilise alors son service d'exécution pour qu'il termine la visite locale de l'agent.

Dans les cas d'un verdict négatif, le service local de migration demande à son service d'exécution de remettre en route l'agent. Mais que ce soit le service de migration local ou celui distant qui relance l'exécution de l'agent, celui-ci se retrouve dans l'état *Post-migration* et il récupère le verdict final par le retour de la fonction *« Migrer() »*.

Ainsi l'agent pourra savoir les raisons d'un éventuel échec ou constater que sa migration s'est bien déroulée et pourra alors adapter son exécution. Par exemple, si le site souhaité est inconnu, il pourra demander une migration vers un autre. Notons que c'est l'agent qui utilise l'interface *« Migrer() »*, il choisit donc lui-même l'instant et la destination de ses déplacements. Un agent est bien proactif.

Notons que si une rupture de la connexion entre les deux services de migration intervient avant de pouvoir exécuter complètement le retour de la méthode, c'est obligatoirement l'agent sur le site d'origine qui est conservé et le clone du site destinataire est automatiquement retiré du système par le service de migration destinataire. Le verdict retenu par le service de migration origine est ***Erreur Connexion***

5.3.4 Le service d'annuaire des agents locaux

Ce service gère une base de données sur les agents qui souhaitent se faire connaître auprès des autres agents présents sur le site. Il s'agit d'un service de pages jaunes classique où les agents peuvent s'inscrire en adressant un message contenant leur identifiant et l'ensemble des propriétés qu'ils souhaitent faire connaître. Nous proposons deux possibilités d'enregistrement.

```
ou.SeFaireConnaître(Agent enVisite)

ou.SeFaireConnaître(Agent enVisite, Liste propriétés)
```

La première permet de faire un enregistrement par défaut où le service d'annuaire récupère lui-même l'ensemble des propriétés de l'agent et la seconde permet au concepteur de diffuser uniquement les propriétés qu'il souhaite partager.

Ce service offre aussi la possibilité d'effectuer des recherches selon différents critères de recherche [KB01] appartenant à l'ensemble des propriétés d'un agent, par exemple ceux qui

veulent coopérer ou encore l'ensemble des agents créés par le même utilisateur. Cette recherche permet de récupérer les identifiants des agents et ainsi de dialoguer avec eux, afin d'engager une coopération par exemple. En cas d'échec, l'identifiant *null* est récupéré.

```
Agent ou.TrouverUnAmi(Chaîne CR)
// Permet de trouver un partenaire

Liste ou.TrouverDesAmis(Chaîne CR)
// Permet de trouver un ensemble de partenaires
```

Note l'abréviation *CR* désigne le critère de recherche dans l'annuaire.

Ce sont les agents qui s'enregistrent volontairement. Il n'y a pas d'obligation. Le service d'annuaire concerne uniquement les agents présents sur le site. Il n'a aucune interaction avec les autres sites. Les agents peuvent aussi se désabonner lorsqu'ils ne souhaitent plus apparaître dans l'annuaire afin de ne plus être référencés par de nouveaux arrivants, lorsqu'ils préparent leur migration par exemple.

```
ou.SeFaireOublier(Agent enVisite)
```

Pour éviter que le service d'annuaire possède des informations incohérentes lorsque des agents migrent sans se retirer de la base par exemple, la fonction *Terminer()* de l'agent intègre obligatoirement l'appel suivant : *ou.SeFaireOublier(this)* . Le service d'exécution utilisant systématiquement *Terminer()* lors des terminaisons de visites, le service d'annuaire possède une base de données cohérente garantie par le retrait de tous les agents partant du site ou se terminant définitivement.

5.3.5 Le service de tableau blanc

Nous mettons en place sur chaque site un service de tableau blanc permettant de déposer et de récupérer des données. Ce tableau permet aux agents en transit de laisser des messages pour des agents qui ne sont pas présents, ou pas accessibles, et qui pourront les récupérer plus tard.

Pour organiser les dépôts, les données sont étiquetées avec l'ensemble des propriétés de l'agent qui dépose. De plus, pour éviter de le surcharger, le service propose un ensemble de critères de stockage, apparaissant dans le service d'annuaire, qui indiqueront aux agents la politique de sauvegarde des informations. Cette politique pourra être définie en fonction de la taille, le temps de stockage, la pertinence des données etc.

```
Liste ou.TBl.CritèreStockage()
```

Lorsque les agents veulent déposer une donnée, ils prennent connaissance des critères de stockage qui leur permettent de choisir la manière d'enregistrer leur information, puis ils créent les données à stocker qu'ils adressent au service de tableau blanc. Cette requête sera composée de trois éléments : du critère de stockage choisi, de la propriété des agents utilisée pour le classement dans le tableau et enfin du contenu lui-même.

```
ou.TBl.StockerInfo( Chaîne critère,
                    Chaîne propriétés,
                    Chaîne données)
```

Les agents peuvent interroger le service de tableau blanc en donnant la propriété cherchée, qui peut être leur identifiant par exemple, puis le service leur envoie l'ensemble de tous les entêtes des données stockées correspondant à la requête et enfin les agents peuvent demander la récupération des informations choisies.

```
Liste ou.TBl.InfoStock(Chaîne critère)

Liste ou.TBl.Recevoir(Liste entête)
```

Nous avons défini ce service de tableau blanc pour la raison principale qu'il nous permet de mettre en place la communication indirecte que nous souhaitions pour les intergiciels, mais aussi, et surtout, les coopérations indirectes nécessaires à la réalisation de la couche d'ambiance.

Nous possédons à présent les agents et les services supports permettant le déploiement d'applications réparties. Nous présentons maintenant le protocole de visite que devra adopter tout agent durant son exécution s'il souhaite être coopératif.

5.4 Protocole de visite d'un site

Nous avons vu dans la section 2.2.1 que la difficulté d'établir des rencontres, surtout dans le cadre de la migration ciblée, impose la mise en place d'une coordination décrivant les différentes étapes respectées par les intervenants lors d'une coopération afin d'en garantir le déroulement intégral et ininterrompu. Cette coordination s'accompagne d'une politique générale de coopération permettant aux agents de garder le contrôle sur l'avancement de leur tâche globale.

Dans cette partie, nous proposons un protocole de contrôle permettant de garantir la coordination et définissant une politique de coopération de base permettant, en particulier, de traiter la terminaison des phases de coopération. Nous présentons, aussi, deux exemples de protocole, ou comportement, permettant de mettre en œuvre différents types de coopération.

5.4.1 Le protocole de contrôle

La coopération entre agents en visite sur un même site nécessite de définir un protocole de synchronisation de ces agents engagés dans une phase coopérative. Un concept de base pour coordonner des actions exécutées par différents processus est celui de conversation proposé par Randell [Ran75]. Cependant, ce protocole, présenté dans le cadre des systèmes tolérants aux fautes, impose des contraintes fortes de synchronisation entre processus afin de garantir en particulier certaines propriétés d'atomicité des actions exécutées dans le contexte d'une conversation. Dans le modèle envisagé, nous adoptons un mécanisme de synchronisation entre paires d'agents afin de conserver un maximum de liberté entre agents.

Lorsqu'un agent mobile arrive sur un site de visite, il doit adopter certaines règles de coopération avec les agents visiteurs déjà présents sur ce site. Il est donc nécessaire de définir un protocole de coopération entre agents.

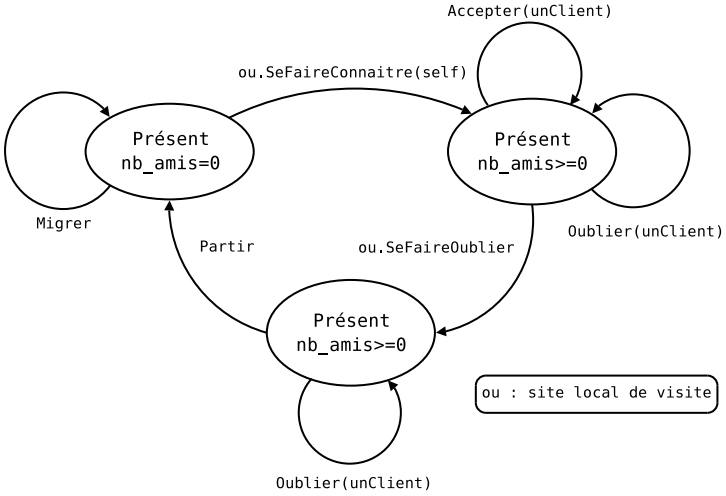


FIG. 5.2 – Protocole de viste d'un agent mobile coopérant

On choisit un protocole dissymétrique de type client/serveur : lorsqu'un agent arrive en visite sur un site, il s'enregistre auprès du service d'annuaire local du site de visite et rentre ainsi dans un état coopératif. Dans cet état, il peut :

- prendre l'initiative de phases coopératives avec d'autres agents eux-mêmes prêts à coopérer ;
- accepter de coopérer avec des agents qui le lui demandent.

Lorsqu'un agent a accepté de coopérer momentanément avec un autre agent visiteur, ce dernier peut invoquer les services offerts par l'agent acceptant. Il se comporte alors comme un client vis-à-vis de cet agent acceptant serveur. Inversement, un agent demandeur (initiateur) ne peut donc utiliser les services d'un agent qu'après acceptation par ce dernier. Nous présentons les différentes possibilités d'évolution de l'agent dans le diagramme de transitions d'états de la figure (5.2).

Nous faisons correspondre le comportement générique d'un agent visiteur, illustré par ce diagramme, avec l'algorithme 5.1 qui devra être exécuté à chaque visite pour garantir la coordination des phases de coopération.

```
ou.SeFaireConnaître(this) ;
// Phase de coopération
ou.SeFaireOublier(this) ;
// état présent mais non coopératif
Partir()
// ⇒ attente fins de phases
//    de coopération
ou.Migrer(Site dest)
```

Algorithme 5.1 - Visite d'un site

L'opération *SeFaireConnaitre(Agent qui)* permet à un agent visiteur de se faire connaître auprès des autres agents présents sur le site et connus du site en s'enregistrant auprès du service annuaire de ce site. Une fois connu, un agent entre dans une phase potentiellement coopérative.

L'opération *SeFaireOublier(Agent qui)* consiste pour l'agent visiteur à s'extraire de l'annuaire du site. Désormais, aucun nouvel agent ne pourra plus coopérer avec lui. Par contre, cet agent doit rester présent jusqu'à ce que tous les autres agents qui coopéraient avec lui l'aient libéré (autorisé à partir).

L'opération *Partir()* contrôle si un agent peut quitter le site. Elle peut donc être bloquante puisqu'il faut attendre éventuellement la fin de phases de coopération auxquelles participait cet agent et qu'il avait acceptées.

Enfin, l'agent pourra alors migrer vers un autre site.

Phase de coopération Dans sa phase coopérative, un agent peut exécuter de façon parallèle, séquentielle ou entrelacée des actions de coopération selon le protocole décrit par l'algorithme 5.2

```
Agent unAmi = ou.TrouverUnAmi(CR);
   // CR = <critère de recherche>
if  (unAmi ≠ null) {
   unAmi.Accepter(this);
   // utiliser les services de l'agent
   // unAmi.m(...); ...;
   unAmi.Oublier(this);    }
```

Algorithme 5.2 - Phase de coopération

Le caractère coopératif des agents est présent dans l'opération *Accepter*. Un agent dans l'état coopératif accepte toujours un nouvel utilisateur. On pourrait bien entendu introduire des conditions spécifiques d'acceptation : identité du demandeur, nombre trop élevé d'agents déjà utilisateurs...

La synchronisation des opérations du protocole de coopération, en l'occurrence le triplet de primitives *Accepter*, *Oublier* et *Partir*, nécessite le comptage des agents (amis) qui ont demandé de coopérer avec l'agent (compteur nb_amis dans la figure (5.2)) La sémantique de ces opérations est décrite par les triplets de Hoare suivants[8] :

$$\begin{array}{llll} \forall k > 0 & \{\textit{coopératif} \wedge nb\_amis = k\} & \text{Accepter(Agent qui)} & \{\textit{coopératif} \wedge nb\_amis = k+1\} \\ \forall k > 0 & \{nb\_amis = k\} & \text{Oublier(Agent qui)} & \{nb\_amis = k-1\} \\ & \{nb\_amis = 0\} & \text{Partir()} & \{nb\_amis = 0\} \end{array}$$

Ce protocole garantit l'absence d'interblocage si et seulement si toute phase de coopération d'un agent est de durée bornée. Dans ce cas, un agent n'accepte plus de nouveaux clients (amis) après l'exécution de la primitive *SeFaireOublier*. Seules des primitives *Oublier* peuvent alors être exécutées par les agents clients (amis). La décroissance du compteur *nb_amis* est ainsi assurée et donc sa convergence vers zéro.

5.4.2 Exemple de comportements coopérants

Nous avons vu qu'un agent, une fois entré dans la phase de coopération, utilise les services proposés par ses homologues. Nous présentons ici un service permettant de mettre en place la coopération directe et un autre permettant de réaliser une coopération indirecte grâce au service de tableau blanc.

[8] Le terme *coopératif* doit être interprété comme un prédicat vrai si l'agent est entré en phase coopérative.

Le protocole de cartographie

Nous présentons un algorithme permettant de réaliser la coopération directe entre deux éléments de la même application, correspondant à la communication de base de notre modèle. Pour cela nous proposons une méthode s'adaptant à notre modèle qui permet d'utiliser le caractère coopérant des agents mobiles pour la cartographie d'un réseau [MKM99]

Le but de cet algorithme est de déterminer, le plus rapidement possible, la topologie d'un réseau inconnu. Pour cela, on définit la tâche principale de l'agent comme l'exploration de tous les nœuds présents dans le réseau en récupérant les sites à visiter auprès du service de localisation et en se déplaçant de site en site, On peut décrire cette tâche comme suit :

- arriver sur un site
- récupérer l'ensemble des identifiants des sites du voisinage
- ajouter les site inconnus à l'ensemble des sites à couvrir
- marquer, dans l'agent, le site en cours comme visité
- si l'ensemble de sites à découvrir est vide alors la tâche est terminée
- sinon choisir un site à explorer et migrer vers lui

Pour permettre une cartographie plus rapide, on utilise plusieurs agents qui vont évoluer chacun de leur côté et vont partager les différentes parties du réseau qu'ils ont découvertes. Chaque agent va alors engager le protocole de coopération de l'algorithme 5.2 avec *CR* = *« Cartographie »* permettant de récupérer un agent (*unAmi*) de même compétence et va exécuter les différentes étapes du protocole décrit par l'algorithme 5.3

```
Liste sitesCouvertsAmi = unAmi.SiteCouverts()
  // Récupère les sites déjà visités
  // par le partenaire
unAmi.AjouterSites(sitesCouverts)
  // Envoie au partenaire les sites déjà
  // visités par l'agent
this.AjouterSites(sitesCouvertsAmi)
  // Ajoute les sites visités par le partenaire
  // à ceux de l'agent
Liste sitesACouvrir = unAmi.sitesACouvrir()
  // Récupère les sites que doit
  // encore visiter le partenaire
this.AjouterSites(sélectionner(sitesACouvrir))
  // Sélectionne les sites qui seront découverts
  // par le partenaire et les considère visités
unAmi.AjouterSites(selectionner(this.sitesACouvrir))
  // Sélectionne les sites à découvrir par l'agent
  // et qui seront considérés visités par le partenaire
```

Algorithme 5.3 - Cartographie coopérative

L'idée ici est de partager le travail effectué entre les agents. L'agent va en fait récupérer les éléments que son partenaire a déjà explorés et il pourra les considérer lui aussi comme

découverts. Puis il regarde les sites que son partenaire connaît sans les avoir encore visités. De cette observation, l'agent détermine les éléments qu'il laissera à la charge de son partenaire et il pourra là aussi les considérer comme visités. Le partenaire récupérant lui aussi les éléments découverts et pris en charge par l'agent.

Cette coopération directe permet en fait de partager les tâches à réaliser entre les agents. En exposant le travail qu'un agent a déjà accompli et celui qu'il va réaliser, il permet à ses partenaires de se concentrer sur leur propre partie du travail qui reste à faire et tous deviennent plus efficaces. Nous avons simplifié la phase de coopération à l'échange entre deux partenaires, mais nous pouvons encore augmenter l'efficacité en faisant coopérer l'agent avec tous les partenaires de l'application présents sur le site.

Le protocole de rumeur

Nous venons de présenter un protocole permettant de réaliser la coopération directe. Nous proposons ici un protocole qui permet de mettre en place la deuxième méthode de communication, *i.e* la coopération indirecte. Nous avons choisi pour cela la réalisation des algorithmes de rumeurs grâce à l'utilisation du service de tableau blanc. Nous emploierons cette méthode pour la réalisation de la *couche d'ambiance* avec les agents légers. Avant de décrire ce protocole, notons que le pré-requis ici est l'application de l'algorithme 5.2 où le critère de recherche (CR) est l'identifiant du service de tableau blanc et par conséquent la variable *unAmi* correspond à l'agent local représentant de ce service.

```
Liste messages = unAmi.InfoStock("rumeurs")
  // Récupère les rumeurs stockées
  // par le service tableau blanc
Liste interresant = this.Analyser(messages)
  // Analyse les entêtes de messages et
  // sélectionne ceux intéressants pour l'agent
Liste rumInterre = unAmi.Recevoir(interessant)
  // Récupère les informations intéressantes
this.AjouterRumTrans(rumInterre)
  // mise à jour des rumeurs transportées
Pour tout éléments i de this.Importantes() faire
     unAmi.StockerInfo("écraser","rumeurs",i)
  // Prépare les rumeurs importantes
  // et les envoie au service de tableau blanc
```

Algorithme 5.4 - Service de rumeur

Le principe de cet algorithme est de récupérer les rumeurs présentes sur le site, de les analyser, de mettre à jour l'agent et enfin d'actualiser le service. L'analyse est à la charge du développeur qui devra proposer une fonction *Analyser* permettant de vérifier la véracité des informations.

L'algorithme se décompose en deux parties duales. La première consiste à récupérer toutes les informations du site, à les analyser, puis à recevoir les rumeurs importantes pour terminer par l'actualisation des données propres de l'agent.

La deuxième partie de l'algorithme fait l'inverse en s'intéressant à la mise à jour du tableau. L'agent sélectionne les informations estimées importantes, c'est toujours le concepteur qui définit le critère, et pour chacune d'elle il envoie un message au service de tableau blanc pour qu'il les mette à jour dans sa base de données, l'agent accepte même l'écrasement des anciennes rumeurs.

Il est intéressant de noter, pour terminer sur cet algorithme de rumeur, que nous avons utilisé simplement la variable *UnAmi* correspondant au service tableau blanc, mais si nous changeons le critère de recherche (CR) par « n'importe qui faisant des rumeurs », nous pouvons appliquer cet algorithme aussi bien à des agents mobiles qu'à un tableau fixe.

5.5 Conclusion

Dans ce chapitre, nous avons présenté notre modèle d'agents mobiles coopérants composé de la définition précise des agents et des services de base que doit offrir le système aux applications. Nous avons défini les agents comme une entité mobile, autonome et proactive qui possède les caractéristiques suivantes :

- un identifiant unique
- un ensemble de propriétés propres
- une localisation précise
- un comportement générique sous le contrôle du système
- une description à partir d'un ensemble de comportements/transitions

Ensuite nous avons proposé un ensemble de services de base permettant aux agents de posséder les éléments nécessaires à leur exécution, leur mobilité et leur communication. Nous avons identifié les services suivants :

- service d'exécution
- service de migration
- service d'annuaire
- service de voisinage
- service de tableau blanc

Cet ensemble de services permet aux agents d'effectuer une visite complète sur un site, allant de l'arrivée jusqu'à la migration, et de rendre possible la prise en compte des rencontres grâce à l'utilisation de l'annuaire qui permet de connaître les partenaires présents et d'engager les coopérations. Pour ces dernières, nous avons ensuite proposé un protocole de visite de base qui garantit la continuité des phases de coopération et assure une politique de coopération minimale laissant l'agent garder le contrôle de son évolution.

Ces comportements généraux de coopération nous permettent de proposer les bases de toute coopération mais ne cherche pas à les limiter. En fait, nous souhaitons proposer un ensemble de comportements fondamentaux que pourront utiliser les concepteurs, comme des agents très coopérants en temps et/ou en volume ou comme des agents très peu coopérants. Les développeurs pourront, selon leurs besoins, utiliser ces comportements par défaut ou en créer d'autres s'ils estiment cela nécessaire. Nous en avons d'ailleurs présenté deux permettant

de réaliser les algorithmes de rumeurs et le partage de tâche appliquée à la cartographie d'un réseau.

Maintenant que nous possédons tous les éléments nécessaires pour réaliser nos applications réparties à base d'agents mobiles, nous allons présenter dans le chapitre suivant plusieurs services applicatifs en commençant par la réalisation de la couche d'ambiance à l'aide des agents légers pour essayer de mesurer, cette fois-ci, la cohérence de l'approche choisie intégrant deux niveau de mobilité.

Troisième partie

Simulations et exemples d'utilisation des agents mobiles coopérants

Services et applications

6

À partir de notre modèle d'agents mobiles coopérants décrit dans le chapitre 5, nous avons réalisé un simulateur en Java, détaillé dans l'annexe A, respectant tous les éléments de base de l'environnement où évoluent les agents. Le but de ce simulateur est de tester notre modèle en mettant en place la double mobilité, décrite dans le chapitre 3, avec l'ensemble des problèmes liés au dynamisme du contexte.

À partir de ce simulateur, nous avons mené un ensemble d'évaluations afin d'expérimenter les solutions que nous avons proposées dans le chapitre 4. En partant de l'étude des résultats, nous allons déterminer l'intérêt de la nouvelle conception des applications réparties, que nous avons proposée, dans les environnements dynamiques à large échelle. Rappelons que cette méthode, basée sur l'utilisation de deux grands types d'agents, doit permettre d'optimiser les déplacements des agents dans un contexte fortement dynamique en proposant une couche d'ambiance réalisée à partir d'un service de localisation. Cette étude, qui constitue la première partie de ce chapitre, porte sur le service de localisation et un service d'équilibrage de charge

Dans une seconde partie, nous présentons deux exemples d'applications se basant sur notre modèle de conception et utilisant la couche d'ambiance. Pour commencer, nous nous intéressons à transposer la programmation par composants dans les environnements dynamiques. Ensuite, nous proposons de transformer une application orientée client, classiquement centralisée, pour qu'elle puisse prendre en compte les unités mobiles. Nous prenons comme exemple une application de gestion de données personnelles en nous focalisant sur l'organisation de rendez-vous.

6.1 Service de localisation

Nous avons vu dans le chapitre 2 que les agents disposent de deux méthodes de coopération, une directe et une indirecte. Dans le modèle présenté dans le chapitre 5, nous avons indiqué que ces deux méthodes sont réalisables lors de la visite d'un agent. La méthode indirecte utilise le service de tableau blanc où un agent peut trouver/déposer des informations et la méthode directe est mise en œuvre grâce à l'utilisation du service d'annuaire local qui permet de mettre en place un schéma de type client serveur entre deux agents. Le service de localisation utilise la coopération indirecte.

La localisation des agents nécessite de rappeler ou de préciser les différentes hypothèses faites sur le réseau dynamique. Nous définissons les propriétés minimales suivantes :

- Tout site s possède un voisinage n_s constitué de l'ensemble des sites qui sont directement accessibles depuis s. Un site dont le voisinage est vide est momentanément isolé (il peut néanmoins rester actif).
- Le mouvement d'un site revient à modifier son voisinage.
- Tout site ne reste pas indéfiniment isolé.
- Il existe des agents dont la mobilité est beaucoup plus importante que celle des sites.

Pour réaliser le service il est nécessaire de nous appuyer sur un ensemble de principes de base. Pour commencer, nous utilisons simultanément des traces dites de visite et des traces dites de voisinage. Nous en donnons la définitions lors de la présentation de la stratégie de migration. Ensuite, nous imposons à tout agent de l'environnement de participer implicitement à une partie de la fonction de localisation, *i.e* nous intégrons aux agents un ensemble d'instructions à exécuter avant toute migration, même à ceux ne faisant pas partie du service de localisation. Puis, nous définissons un ensemble d'agents légers dédiés spécifiquement à la fonction de localisation. Enfin, les agents coopèrent de façon indirecte via le service de tableau blanc des sites visités.

Pour mettre en place ces principes, nous avons besoin de définir le service de localisation sur une base de données locale stockant les traces, sur le service d'annuaire d'agents locaux et sur une stratégie de migration adaptée. Une des caractéristiques de l'approche est de combiner à la fois les fonctions de localisation et de rencontre des agents

6.1.1 La stratégie de migration adaptative

Une stratégie de migration adaptative est un méthode permettant de guider un agent lors de ses phases de migration. Elle s'apparente à un routage classique, excepté que les conditions de routage sont à un niveau purement applicatif. Son objectif est de minimiser le nombre de sauts réalisés par l'agent pour atteindre un partenaire particulier appelé agent $cible$. Deux approches sont possibles pour ce type de routage : soit réactive lorsqu'un chemin est construit à la demande, soit proactive lorsque le protocole met à jour de façon continue les informations de routage [BDDN01].

Notre modèle de schéma de migration est basé sur la coopération indirecte en exploitant la notion de trace d'agents mobiles. L'idée de ce mécanisme de coordination est inspirée de la théorie de la stigmergie, développée par le biologiste Grassé [Gra59] au sujet d'une colonie de fourmis. Les fourmis ne communiquent pas directement entre elles ; les échanges d'informations passent par une modification de l'environnement pour réaliser des tâches complexes comme la répartition de charge [BDGO03].

En utilisant la coopération indirecte, les agents ne coopèrent pas directement entre eux afin de trouver leurs partenaires cibles. Ils profitent des traces laissées par les autres agents sur les sites visités. En utilisant cette méthode, nous garantissons l'indépendance des agents nécessaire à la prise en compte de la double mobilité (cf section 4.3) grâce à la garantie de la totale autonomie de chaque agent.

Une trace est une information laissée par un agent, sur un site visité, en vue d'aider un autre agent à retrouver sa cible plus rapidement. Cette information peut être soit la direction à suivre par un agent pour atteindre sa cible, soit les agents rencontrés lors des récentes visites d'agents

tiers, soit un endroit spécifique où l'agent cible a été aperçu ou encore toute autre information similaire.

Pour le service de localisation nous identifions deux types de trace. Le premier concerne la nouvelle direction de migration d'un agent, ce que nous désignons par *trace de visite*. Lorsqu'un agent termine la visite d'un site et qu'il s'apprête à migrer, il laisse la trace du site destination où il va entamer sa prochaine visite. Ceci constitue l'équivalent d'un lien de poursuite pour le routage de messages. Cette trace constitue la participation minimale, et obligatoire, que se doit d'acquitter chaque agent auprès du service de localisation. Pour cela, nous intégrons à la méthode `Terminer()` de tout agent, l'appel à la fonction `ou.aiguilleur.Enregistrer(<this>,<vide>,dest)` permettant d'enregistrer pour l'agent en fin de visite (`this`) la trace de sa destination immédiate (`dest`).

Le deuxième type de trace concerne la direction où un agent a été « aperçu ». Ce type de trace peut être assimilé à la propagation de rumeurs [D+87]. Nous désignons ce type de trace par *trace de voisinage*. Les traces de voisinage fournissent la localisation des agents récemment rencontrés par tout agent visiteur d'un site. Il ne s'agit pas forcément des agents rencontrés dans le voisinage immédiat. Plus précisément, pour un agent visiteur Av, un site S_a est dit dans le voisinage de niveau p d'un site S_b, si le chemin parcouru par Av du site S_a au site S_b est de longueur égale à p. Cette longueur s'exprimant en nombre de migrations, $p = 3$ implique que trois migrations sont nécessaires à l'agent Av pour aller du site S_a au site S_b. Pour un site S_a donné, la trace de voisinage de niveau p, désigne l'ensemble des agents rencontrés dans le voisinage de niveau p de A.

Pour une trace de voisinage de niveau p, un agent se déplace avec une copie de la liste des ensembles d'agents $AR_p, AR_{p-1}, \ldots, AR_1$ qu'il a rencontrés sur chacun des p derniers sites visités. Après chaque migration, l'information transportée est enregistrée sur le site d'arrivée et mise à jour par décalage des éléments de la liste : AR_p est oubliée et AR_1 mémorise les agents présents sur le site courant. Soit AR_i^p l'ensemble des agents rencontrés dans le voisinage de niveau p du site i, alors $T_i = \{\bigcup_{k=1}^{k=p} AR_i^k\}$ représente la trace de voisinage enregistrée sur le site i par l'agent en cours de visite.

Les traces de visite et de voisinage sont dans la même base de données de traces gérée par le service de tableau blanc. Ainsi, à tout instant, un agent trouve au plus une seule trace sur un site donné, le principe retenu étant celui de la trace de visite ou de voisinage la plus récemment enregistrée sur le site en fonction du niveau p. Il aurait été plus précis de dater les traces, mais cela engendre le problème de la gestion d'un temps global, problème bien connu dans des environnements distribués. Cet aspect sort du cadre de ce travail, mais nous verrons lors de l'étude des simulations qu'il a un effet non négligeable pour l'efficacité du service de localisation.

À cause de la mobilité physique des sites, une trace peut être obsolète si elle pointe un site qui n'est plus voisin du site courant. Pour avoir une base de données de traces plus cohérente, le service de localisation interroge régulièrement le service de voisinage, qui donne les éléments atteignables dans le voisinage d'un site, et peut donc vérifier la validité d'une trace lors de leur usage. Notons qu'il ne s'agit pas uniquement des traces de niveau $p = 1$ mais bien de toute trace indiquant un itinéraire qui ne peut être atteint depuis le site.

6.1.2 Interface du service de localisation

Pour offrir aux agents un accès homogène et direct au service de localisation présent sur chaque site, nous définissons la référence unique *ou.aiguilleur* correspondant à l'interface du service. La première primitive composant cette interface concerne la base locale de traces de visite.

```
Liste ou.aiguilleur.Enregistrer(Liste<AgentId> agentsVoisins,
                                Liste rumeurs
                                Site origine)
```

Cette fonction permet de mettre à jour la base locale de traces grâce à la liste *agentsVoisins* correspondant à l'ensemble des agents présents sur le site *origine*, celui de la précédente visite, et aussi grâce à la liste *rumeurs* correspondant aux autres agents rencontrés durant les précédents déplacements. La fonction renvoyant une liste de rumeurs mises à jour.

Notons que pour la gestion de la base de données de traces, le service de localisation utilise le service de tableau blanc et que, pour obtenir l'ensemble des agents présents sur le site, il s'adresse au service d'annuaire.

Pour les agents réalisant le service de localisation, l'appel à la primitive *Enregistrer* est intégrée directement à la méthode *Arriver()* d'un agent qui est utilisée à chaque nouvelle visite. Pour permettre de stocker la liste retournée par la méthode *Enregistrer* qui correspond à l'ensemble des agents présents sur le site de visite, nous avons besoins que les agents disposent d'une primitive permettant de construire l'ensemble des traces à propager. Pour cela, nous ajoutons une primitive à l'interface des agents participant intégralement à la localisation. Elle sera aussi utilisée par la fonction *Arriver()*. Il s'agit de la primitive *AjoutTraces()*

```
unAgent.AjoutTraces(Liste agentsPresents)
```

Les agents gèrent donc une liste interne de traces qu'ils véhiculent avec eux et qu'ils propagent sur les sites visités. Maintenant que nous disposons des primitives permettant la propagation des traces, nous devons proposer aux agents un moyen de connaître la localisation d'un partenaire. Pour cela nous utilisons une primitive du service d'annuaire et une primitive propre au service de localisation.

La primitive du service d'annuaire utilisée est *TrouverUnAmi(Chaîne CR)* permettant de trouver un agent sur le site. Ici le critère *CR* est l'identifiant de l'agent $cible$ ce qui permet de tester (et donc localiser) si le partenaire visé est bien présent sur le même site que l'agent $demandeur$. Si cette primitive échoue, elle renvoie une référence $null$ et l'agent demandeur va alors consulter le service de localisation pour connaître les traces.

```
Agent ou.TrouverUnAmi(identifiantCible)
```

La seconde primitive, propre au service de localisation, est *aiguilleur.Guidage()*. Elle permet de demander vers quel nœud, **voisin** du site de visite courant, un agent $demandeur$ doit migrer pour rencontrer un agent $cible$. La primitive ne renvoie que le site voisin vers lequel le demandeur devra migrer pour tenter d'atteindre l'agent cible dont l'identifiant global est fourni en paramètre. Le service de localisation ne donne pas directement la localisation de l'agent cible mais simplement un début d'itinéraire à suivre. La primitive assure donc une fonction de guidage vers l'agent cible.

`Site ou.aiguilleur.Guidage(String indentifiantCible)`

Le service d'annuaire local, hébergé par chaque site du réseau, enregistre seulement les agents présents sur le site à un instant donné. Tout agent arrivant sur un site et souhaitant se faire connaître, s'enregistre dans l'annuaire local du site. Un agent qui cherche à rencontrer un partenaire consulte donc l'annuaire local pour retrouver sa cible (appel à `ou.TrouverUnAmi`). Si celle-ci n'est pas présente, alors l'agent migre vers un autre site, possédant un autre annuaire, suivant la stratégie de migration (guidage) mise en place par appel de la primitive `ou.aiguilleur.Guidage`. Le principal avantage de l'annuaire décentralisé proposé est que les mises à jour et les consultations sont purement locales. Elles ne demandent aucune communication à distance, ce qui permet d'éviter les problèmes de rupture de communication due à la double mobilité et permet de ne pas engendrer une surcharge supplémentaire dans un réseau généralement avare en bande passante.

Quand un agent cherchant une cible entame une nouvelle visite, il adopte toujours le même comportement lui permettant d'exploiter les traces grâce à l'interface décrite ici. Pour cela, l'agent consulte l'annuaire local en utilisant la primitive `Ou.TrouverUnAmi(cible)` et si sa cible est présente il obtient son identifiant permettant d'engager une coopération directe. Dans le cas contraire où la cible n'est pas présente localement (retour le l'identifiant `null`), l'agent interroge le service de localisation pour obtenir la trace de sa cible en appelant la méthode `ou.aiguilleur.Guidage(cible)`. S'il existe une trace de la cible, l'agent choisit de migrer vers le site fourni par la trace. La trace exploitée est consommée, effacée. Cette stratégie limite le nombre de traces obsolètes.

Sans trace de la cible, l'agent doit alors choisir un nouveau schéma de migration, qui peut être aléatoire, ou attendre qu'une trace soit disponible. Le risque de famine, si deux agents se poursuivent, est minimisé dans la mesure où les traces de voisinage peuvent créer des raccourcis favorables au poursuivant. Les mises à jour et la consultation du service de localisation ne demande pas de communication à distance, ceci permet de ne pas avoir d'impact majeur sur le coût global de l'application et d'augmenter sa tolérance aux fautes en supprimant les risques de rupture de communications distantes.

À tout instant, l'ensemble des annuaires locaux de chaque site indique la position effective des agents dans le réseau. L'exploitation des traces des agents permet un guidage partiel vers leur annuaire de localisation en combinant les traces de visite et de voisinage. Nous allons à présent évaluer l'apport de la stratégie de localisation adaptative grâce à l'expérimentation qui compose la section suivante.

6.1.3 Expérimentation et évaluation

Pour tester le service de localisation, nous avons utilisé le simulateur présenté dans l'annexe A. Le but des simulations est d'évaluer la performance globale des rencontres entre agents et de percevoir l'impact de la mobilité des sites sur la stratégie de routage choisie. Il faut noter que la performance d'une application à base d'agents mobiles est déterminée en terme de coût induit par la migration des agents.

La mobilité du réseau est simulée en modifiant périodiquement les voisins immédiats de chaque site en gardant constant un nombre moyen de voisins à 6 (densité du réseau de 3). Le principe de la simulation est de comptabiliser, d'une part l'ensemble des migrations effectuées par les agents $\sigma_m(t)$ et d'autre part l'ensemble des rencontres réalisées par ces agents $\sigma_r(t)$, ceci pour calculer la moyenne $M(t)$ du nombre de migrations sur une durée donnée t. Cette

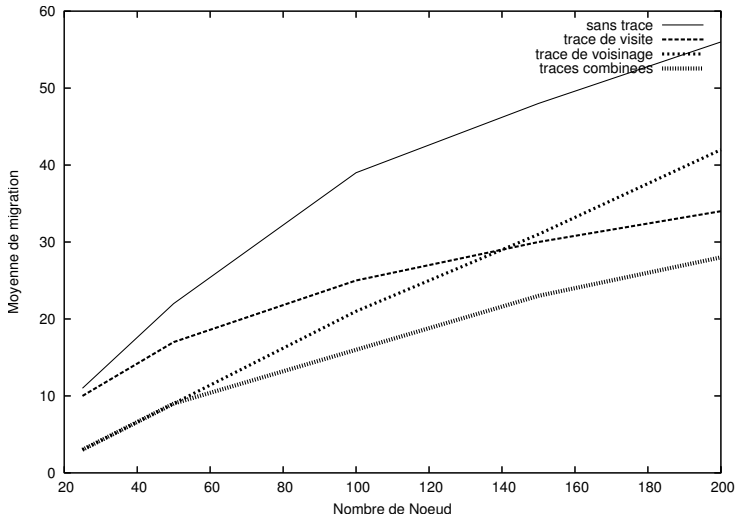


FIG. 6.1 – Impact du mécanisme de routage par trace de visite et de voisinage

moyenne représente le nombre moyen de migrations effectuées par un agent pour faire une rencontre, c'est-à-dire atteindre sa cible.

$$M(t) = \frac{\sigma_m(t)}{\sigma_r(t)}$$

En fonction de la moyenne des migrations calculée suivant différents paramètres de simulation, nous pourrons évaluer si le protocole proposé permet aux agents de se rencontrer plus efficacement, c'est-à-dire avec moins de migrations par rencontre. En modifiant la rapidité de mouvement des sites, nous pouvons, par ailleurs, mesurer l'impact de cette mobilité sur la stratégie.

Avant de la détailler dans l'annexe A, notons que la plate-forme de simulation est composée, entre autres, des principaux services suivants :

- un service de supervision chargé de la supervision des rencontres entre agents (comptage des migrations et des rencontres) et du contrôle de la durée de la simulation.
- un service de gestion de la mobilité des sites chargé du changement de voisinage des sites du réseau. Le nombre moyen de voisins immédiats de chaque site est un paramètre de la simulation (ici, 6 voisins pour les résultats présentés).
- un service de gestion des agents mobiles qui gère le cycle de vie des agents, y compris leurs migrations.

Évaluation du service de localisation

Pour commencer nos expérimentations, nous avons évalué le service de localisation avec uniquement des agents participant entièrement à la localisation et avec quatre schémas de migrations différents pour faire une rencontre : migration aléatoire, migration guidée par les traces de visite, migration guidée par les traces de voisinage et migration guidée par les traces

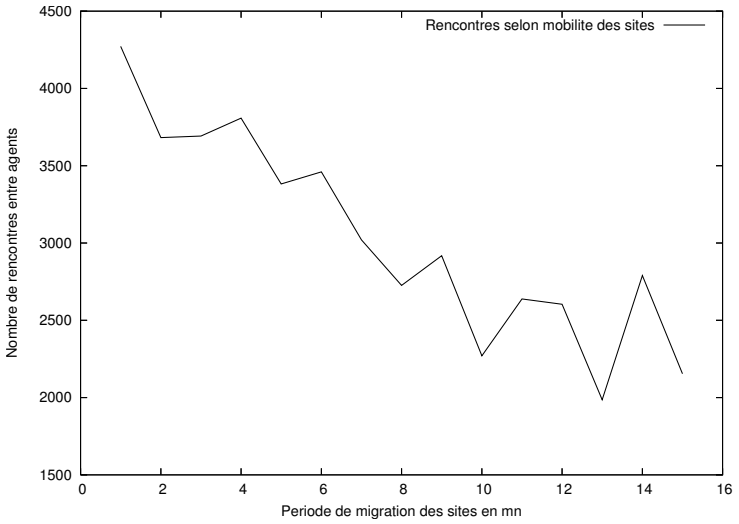


FIG. 6.2 – Impact de la mobilité des sites sur la localisation des agents

combinées. Pour les traces de voisinage, nous nous sommes limités au voisinage de niveau 3, les résultats étant sensiblement les mêmes lorsqu'on va au delà.

Dans chaque cas, nous faisons varier le nombre d'agents de 25 à 200 et le nombre de sites de 25 à 200. La périodicité de changement de voisinage des sites et la durée de visite d'un agent ont été d'autres critères de simulation en conservant cependant le même rapport entre la mobilité des sites et celle des agents (agents 100 fois plus mobiles) et le nombre moyen de voisins. Nous prenons comme référence, le schéma de migration aléatoire où l'agent se déplace de façon aléatoire pour rencontrer sa cible. Nous comparons par la suite, l'effet de la migration guidée à celui du déplacement aléatoire.

Les résultats obtenus sont présentés dans la figure 6.1. L'utilisation des traces de visite permet de diminuer la moyenne des migrations par rencontre ($M(t)$) de l'ordre de 40 %, surtout dans le cas où le nombre de sites du réseau devient important. Pour 200 agents et 200 sites, la moyenne dans le cas de migrations aléatoires est de 56 migrations. Cette moyenne baisse à 34 migrations lorsque l'on utilise seulement les traces de visite et à 42 lorsqu'on utilise seulement les traces de voisinage. Chaque type de trace améliore (diminue) la moyenne de l'ordre de 25 %.

L'exploitation simultanée des deux types de traces augmente l'efficacité du guidage, puisqu'elle permet de doubler le nombre moyen de rencontres sur une période fixée. Dans le cas d'une migration aléatoire, pour 200 agents et 200 sites, la moyenne M est 56 migrations pour une rencontre, alors que pour les mêmes données, cette moyenne diminue jusqu'à 28 migrations par rencontre quand on exploite les traces combinées.

La figure 6.2, montre que le nombre de rencontres diminue avec la mobilité du réseau ; moins les sites bougent rapidement, plus le nombre de rencontres décroît. Mais cette mobilité n'a pas d'impact majeur sur la moyenne M.

Ces résultats montrent que le service de localisation, couplé avec la migration guidée, améliore la performance des agents mobiles en minimisant leurs déplacements. Les tests ont été faits dans un environnement mono-processus Unix. Chaque agent étant représenté par un thread, les simulations ne peuvent dépasser quelques centaines d'agents. Une extension répartie permettrait des simulations à plus large échelle.

Évaluation de la couche d'ambiance

La deuxième expérimentation que nous avons menée porte sur l'évaluation du service de localisation en tant que couche d'ambiance. Celle-ci devant offrir une représentation plus stable de l'environnement aux applications métiers réalisées avec des agents lourds. Pour évaluer cette couche, nous avons réalisé trois types de simulations.

La première consiste à mettre en place des agents lourds sans aucun mécanisme de guidage lors des phases de migration, ils se déplacent au hasard pour rencontrer leurs partenaires, nous l'appelons *Hasard*. Pour la seconde, nous intégrons le service de migration directement dans les agents lourds en utilisant la méthode de surcharge décrite dans la précédente expérimentation, nous la notons *Surcharge*. Enfin pour la troisième simulation, les agents lourds participent au minimum à la fonction de localisation, *i.e* par les *traces de visite*, en utilisant la stratégie de guidage donnée par la couche d'ambiance qui réalise le service de localisation. Celui-ci est alors construit sur un ensemble d'agents légers participant uniquement à la fonction de localisation, nous notons cette méthode par le terme *Ambiance*. Notons que les agents légers s'occupent uniquement de propager les rumeurs, ils sont totalement indépendants et ne cherchent jamais à engager de coopération directe. *i.e* ils utilisent uniquement le service de tableau blanc, ne s'enregistrent jamais auprès du service d'annuaire (ils sont anonymes) et surtout n'appelle jamais la primitive `aiguilleur.Guidage()`, évitant ainsi toute consommation inutile de traces afin de les réserver aux agents lourds (applicatifs).

Pour ces trois simulations, nous avons défini le comportement générique d'un agent lourd comme suit :

- D'abord, il choisit aléatoirement le partenaire avec lequel il souhaite coopérer
- Ensuite, il interroge le service de migration afin de connaître la direction à suivre pour atteindre a cible
- Enfin, quand il arrive sur le site de son partenaire, il coopère avec lui durant le temps minimal d'utilisation du service réalisé par le partenaire.

Notons qu'il existe une différence de comportement lors de l'appel au service de localisation entre les méthodes *Surcharge* et *Ambiance*. Lorsqu'un agent demande un guidage et que celui-ci ne peut lui être fourni, il effectuera alors une migration au hasard dans le cas de la *Surcharge* et par contre, pour *l'Ambiance*, il attendra sur le site en cours tant qu'une trace de guidage ne lui sera pas communiquée par le service de localisation.

Rappelons que le principe de la simulation est d'obtenir, sur une durée donnée t, le nombre moyen de migrations $M(t)$ nécessaire à un agent pour atteindre sa cible à partir des migrations effectuées $\sigma_m(t)$ et des rencontres réalisées $\sigma_r(t)$.

Les implémentations des méthodes de *Surcharge* et d'*Ambiance* sont comparées entre elles et à la méthode *Hasard* selon les critères suivants :

- la surcharge du réseau, *i.e* le nombre moyen de migrations par rencontre
- la moyenne de rencontre, *i.e* le temps moyen nécessaire à la réalisation d'une rencontre
- la stabilité, *i.e* la variation des moyennes en fonction de la taille du réseau.

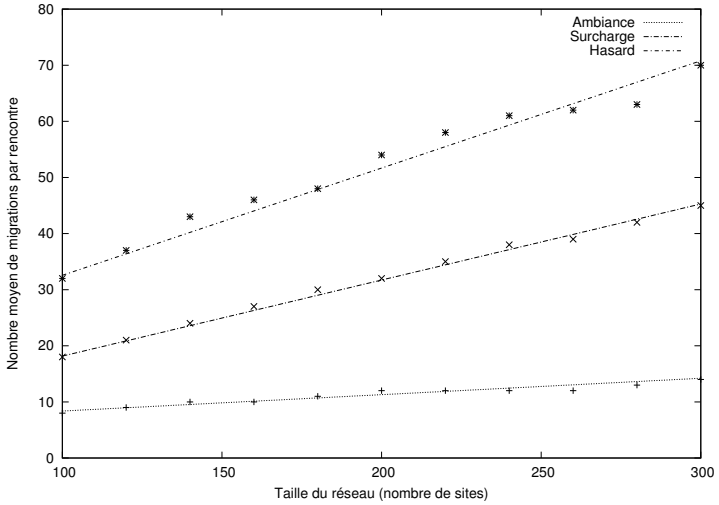


FIG. 6.3 – Comparaison de la surcharge du réseau

Nous avons fixé un nombre moyen de voisins par site (à savoir 4) et nous avons considéré les paramètres suivants : le nombre de sites, le nombre d'agents lourds et le nombre d'agents légers intégrés au service de localisation. Le nombre de sites, *i.e* la taille du réseau, est compris dans l'intervalle $[100, 300]$. Les simulations ont été réalisées avec 100 agents lourds et 15 agents légers. La mobilité des agents légers étant fixé à 1000 fois plus fréquente que celle des sites

Sur la figure 6.3, nous voyons que les résultats montrent une amélioration des méthodes de guidage, par rapport à la méthode au hasard, grâce à la diminution de l'utilisation du réseau et ce, particulièrement, pour la méthode d'ambiance. En effet, le guidage de surcharge permet déjà de diviser par **deux** le nombre de migrations nécessaires pour une rencontre et la méthode d'ambiance, quant à elle, permet une division par **cinq** de ce nombre. L'utilisation de la couche d'ambiance permet donc de réduire de manière significative le nombre de déplacements des agents lourds.

Nous pouvons voir aussi que l'utilisation de la couche d'ambiance permet aux agents lourds d'être quasiment insensibles à la taille du réseau. Ici, avec une taille du réseau passant du simple au triple ($100 \cdots 300$ nœuds), le nombre de migrations nécessaires pour rencontrer un partenaire cible reste pratiquement stable alors que les deux autres méthodes font plus que doubler leur nombre initial. Notons que le nombre moyen de migrations semble être très proche du diamètre du réseau. Ceci nous permet de constater que la couche d'ambiance est mieux adaptée aux environnements envisagés, *i.e* dynamiques et à large échelle physique, que les autres méthodes de rencontres.

Durant nos expérimentations, nous avons aussi cherché à savoir si le nombre d'agents lourds évoluant dans l'environnement pouvait avoir une influence sur l'efficacité de la couche d'ambiance. Pour cela, nous avons réalisé une simulation en fixant la taille du réseau et le nombre d'agents légers tout en faisant varier le nombre d'agents lourds. Cette variation permet de constater l'évolution de la moyenne de migrations par rencontre et donc l'influence du nombre d'agents lourds dans l'environnement.

Les résultats obtenus, présentés sur la figure 6.4. montrent que le nombre d'agents lourds présents dans l'environnement n'a pas une grande influence sur la moyenne des rencontres. Elle reste quasiment stable à plus ou moins une migration. L'efficacité de la couche d'ambiance

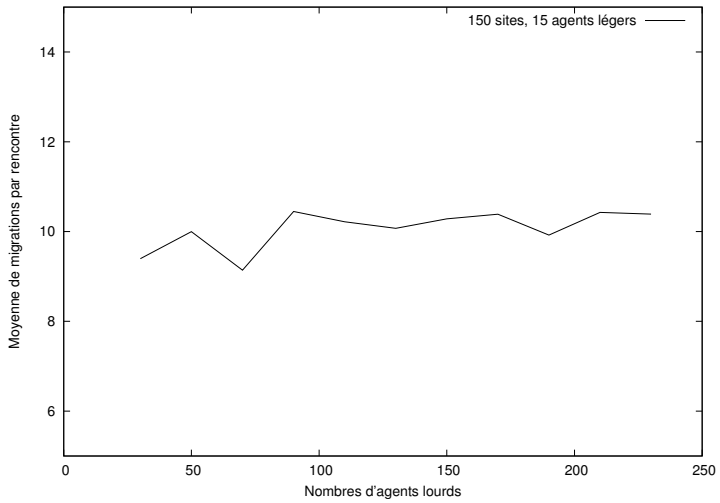


FIG. 6.4 – Influence du nombres d'agents lourds

restant constante, nous pouvons donc constater qu'elle s'adapte aussi aux changements dynamiques du nombre d'agents applicatifs présents dans l'environnement et par conséquent elle s'accommode aussi d'une large échelle au niveau des agents applicatifs.

Évaluation du temps de réponse

Pour compléter les analyses réalisées sur l'utilisation du service de localisation en tant que couche d'ambiance, nous nous sommes intéressés au temps moyen nécessaire à un agent pour atteindre son partenaire cible. Nous avons donc mesuré la moyenne horaire de rencontres opérées par les agents en fonction des trois approches pour la localisation. Les résultats sont présentés sur la figure 6.5.

Nous retrouvons sur ces trois courbes, une pour chaque méthode, l'évolution de la moyenne horaire de rencontres, effectuées par les agents, en fonction de la taille du réseau. Nous pouvons constater que la méthode de surcharge donne les meilleurs résultats. Nous pouvons l'expliquer par le choix fait pour le comportement des agents lourds lorsqu'ils interrogent la couche d'ambiance pour obtenir la politique de guidage. En effet, dans le cas où aucune trace ne peut-être fournie au demandeur, l'agent attend jusqu'au moment où le service de localisation soit en mesure de satisfaire sa requête, puis il entame enfin son déplacement. À l'inverse, dans le cas de la surcharge, lorsque un demandeur ne peut récupérer une direction explicite, il se déplace au hasard et ne perdra pas de temps dans une attente passive. Cette différence de comportements explique une partie de l'écart entre les deux méthodes de guidage.

Mais pour bien évaluer le temps de réponse nécessaire de chaque approche, il faut mettre en corrélation le temps moyen et le nombre de migrations nécessaires à un agent pour atteindre son partenaire. Cette corrélation est exposée sur le tableau 6.1. Nous pouvons constater, pour commencer, que l'utilisation de la méthode au hasard ne se justifie en aucun cas car elle demande naturellement plus de migrations et plus de temps pour atteindre une cible.

Ensuite, nous voyons que la méthode de guidage par surcharge reste plus rapide pour trouver un partenaire, mais nécessite quasiment le double de migrations par rapport à l'utilisation de la couche d'ambiance. Nous avons expliqué une partie de cet écart de temps à cause des

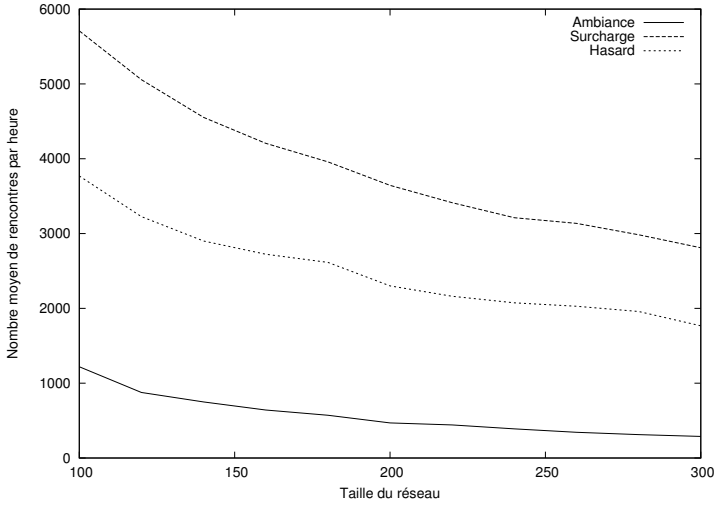


FIG. 6.5 – Comparaison de la moyenne de rencontre

Implémentation	Temps de réponse	Surcharge réseau
Hasard	1.56 sec.	56 migrations
Surcharge	1 sec.	35 migrations
Ambiance	7.80 sec.	12 migrations

TAB. 6.1 – Temps moyen de rencontre et de migration

différences de comportements des agents lourds entre la méthode de surcharge et celle d'ambiance. Un autre phénomène est aussi à prendre en compte, celui de la faible bande passante disponible. Bien que nos simulations intègrent les temps de transmission entre deux sites, nécessaires lors de la réalisation d'une migration, la bande passante théorique dont nous disposons ici reste illimitée alors que cela n'est pas le cas dans la réalité. En effet, avec les limites de la bande passante existantes dans les architectures hybrides, les temps de transmission pourront s'accroître dans le cas d'un réseau surchargé et ainsi augmenter le temps de réponse final si le nombre de migrations est important. Ce phénomène devrait réduire les écarts entre les méthodes de guidage.

Mais même si cet écart devrait se réduire dans un environnement réel, cette différence va demander aux concepteurs de bien identifier les besoins de leurs applications afin de savoir s'ils devront minimiser les temps de réponse ou les migrations. Ils pourront ainsi choisir la méthode de guidage à utiliser. En fait, le choix de la méthode guidage va correspondre avec celui portant sur le type d'agents à utiliser, choix à faire lors de la conception. Le concepteur devra donc définir des agents se déplaçant plutôt rapidement et fréquemment, se rapprochant ainsi des agent légers, ou se déplaçant peu souvent et lentement, plus proches des agents lourds.

Évaluation de la configuration de la couche d'ambiance

Pour la dernière série d'expérimentations que nous avons menées sur la couche d'ambiance, nous nous sommes intéressés au dimensionnement que doit avoir le service de localisation par rapport à la taille du réseau. Le but est de trouver le rapport optimal afin d'obtenir le

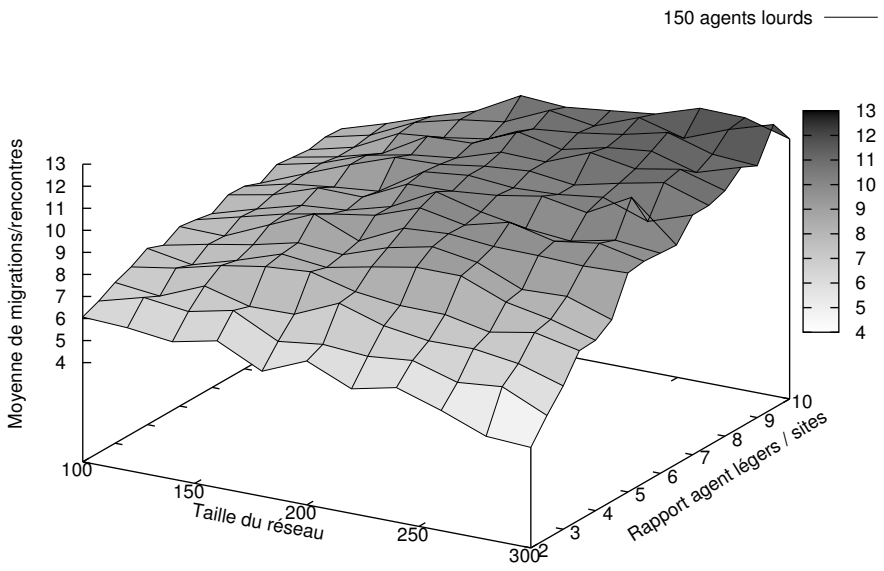


FIG. 6.6 – Adéquation de la couche d'ambiance

meilleur guidage possible pour les agents lourds. En fait, le service étant rendu par un ensemble d'agents légers, nous cherchons le meilleur rapport entre le nombre d'agents légers composant la couche de localisation et le nombre de sites constituant le réseau. Rappelons que le nombre d'agents applicatifs présents dans l'environnement n'a pas d'effet sur l'efficacité de la couche d'ambiance (fig 6.4) et qu'il constituera un paramètre constant fixé à 150 agents lourds pour toutes les simulations.

Pour obtenir le rapport optimal, nous avons réalisé une série de simulations en faisant varier deux paramètres et en récupérant deux types de résultats. Le premier paramètre variable est la taille du réseau comprise dans un intervalle de $[100 : 300]$ sites. Le deuxième est un pourcentage d'agents légers en fonction du nombre de nœuds, nous l'avons fait varier entre 2% et 10%. Pour les résultats, nous partons du même principe que les précédentes expérimentations en nous basant sur une durée donnée t pour obtenir, d'une part, le nombre moyen de migrations $M(t)$[9] et le nombre total de rencontres $R(t)$, d'autre part.

Commençons par regarder les résultats des simulations pour la moyenne $M(t)$ présentés sur la figure 6.6. Étonnamment, nous pouvons constater qu'un grand nombre d'agents légers n'est pas une garantie de performance pour la couche d'ambiance. Nous pouvons le voir sur les deux types de variations. Premièrement si on considère la variation de la taille du réseau, un faible pourcentage (2%) permet d'augmenter l'éfficacité de la couche d'ambiance quand le nombre de sites augmente, alors qu'avec un pourcentage moyen (5%) elle reste stable et qu'avec un fort pourcentage (10%) l'efficacité diminue nettement. Deuxièment si on considère la variation du pourcentage, on constate qu'avec n'importe quel nombre de sites, l'augmentation du pourcentage d'agents fait diminuer l'efficacité du guidage. Pour 200 sites par exemple, la moyenne de migrations par rencontre passe de 6 pour 2%, a 11 pour 10%.

[9]Rappel : $M(t)$ s'obtient à partir des migrations effectuées $\sigma_m(t)$ et des rencontres réalisées $\sigma_r(t)$ nécessaires à un agent pour atteindre sa cible

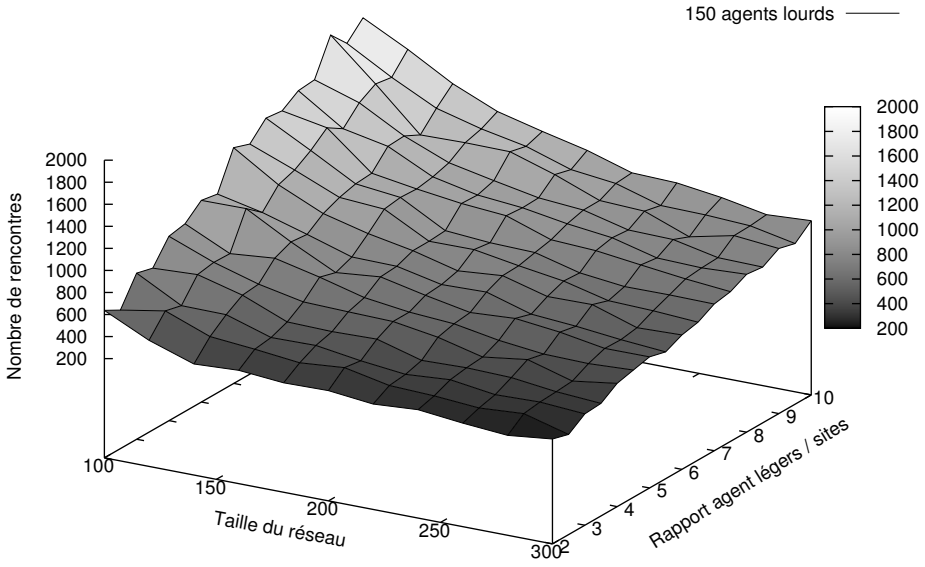


FIG. 6.7 – Adéquation de la couche d'ambiance - suite

Nous pouvons expliquer cette perte d'efficacité lorsque le nombre d'agents légers augmente par la présence de rumeurs redondantes. Nous avons vu que la véracité des rumeurs est gérée à partir du nombre de migrations p permettant à un agent lourd d'atteindre sa cible. Ne disposant pas d'une horloge système globale, nous considérons que la trace de voisinage la plus récente correspond à celle dont le nombre p est le plus petit. Or si nous prenons deux traces (t_1, t_2) de niveaux (p_1,p_2) tel que $p_1 > p_2$ et que t_1 soit plus récente que t_2, le service de localisation va garder la trace t_2 comme référence alors qu'il aurait fallu garder t_1. En gardant la mauvaise trace de voisinage, les agents lourds vont obtenir une politique de migration erronée et augmenter le nombre de déplacements nécessaires pour atteindre un partenaire. Ce phénomène est réduit avec un faible nombre d'agents légers mais va s'accroître si leur nombre augmente. Ceci explique la perte d'efficacité constatée.

Passons maintenant aux résultats du nombre de rencontres $R(t)$ présenté par la figure 6.7. Logiquement, nous constatons qu'un grand nombre d'agents légers permet d'augmenter significativement l'efficacité de la couche d'ambiance. Nous pouvons là aussi le voir sur les deux types de variations. Premièrement en regardant le taux d'agents légers fixé à 2%, nous pouvons remarquer que le nombre de rencontres est approximativement divisé par un facteur 2.5 lorsque la taille du réseau est triplé, montrant ainsi qu'un faible taux d'agents ne permettra pas d'obtenir une efficacité satisfaisante dans les environnements à large échelle. Il est intéressant de noter que ce facteur de baisse se réduit avec un nombre plus élevé d'agents (avoisinant les 2.2 pour un taux de 10%). Deuxièment si on regarde la taille du réseau fixée à 200 nœuds, on constate que le nombre de rencontres est multiplié par 3.5 lorsque le taux d'agents légers passe de 2% à 10%, ceci montre l'intérêt d'avoir une forte concentration d'agents légers. Notons que ce facteur multiplicateur garde à peu près le même ordre de grandeur lorsque la taille du réseau change.

Nous expliquons facilement le gain d'efficacité de la couche d'ambiance lorsqu'elle est composée d'un grand nombre d'agents légers par une meilleure propagation des rumeurs. En effet, lorsque les agents sont peu nombreux, les rumeurs auront du mal à être diffusées sur l'ensemble du réseau, ce qui provoque de fréquents blocages des agents lourds ne pouvant récupérer de politique de guidage. Avec un nombre plus conséquent d'agents, les rumeurs sont plus largement répandues limitant ainsi les temps d'attente et donc permettant d'augmenter le nombre de rencontres totales, *i.e* la rapidité des rencontres ciblées.

À partir de l'analyse de ces deux graphiques, nous pouvons conclure qu'il n'existe pas de taille idéale pour la couche d'ambiance permettant à la fois de maximiser le nombre de rencontres et de minimiser le nombre moyen de migrations. En effet, si nous raisonnons en terme d'efficacité, d'un point de vue de la moyenne de migrations, il est intéressant de diminuer les nombres d'agents légers alors qu'il faut l'augmenter pour le nombre de rencontres. Nous devons donc faire un compromis entre ces deux facteurs qui permet d'avoir un nombre de rencontres satisfaisant et une moyenne de migration le plus proche possible de la stabilité. En regardant les deux figures, nous pouvons voir que le taux répondant le mieux à ces deux contraintes est aux alentours de 5%. Les resultats de ce taux indiquent une moyenne de migrations oscillant à plus ou moins une migration, entre 7.5 et 9.5, et un nombre moyen de rencontres aux alentours de 730.

Cependant, pour pouvoir garder ce taux constant, nous devons être capables de récupérer dynamiquement des informations globales sur l'environnement. Dans notre cas il s'agit du nombre de nœuds appartenant à l'architecture ainsi que le nombre d'agents composant la couche d'ambiance. Nous étudions ce problème dans la section suivante (6.2) en l'appliquant à la régulation de charge. Enfin, pour optimiser la régularisation du nombre d'agents de la couche, il serait intéressant de disposer de mécanismes permettant de créer/détruire dynamiquement des agents et surtout de contrôler l'évolution de la taille afin d'éviter des créations ou des destructions inutiles voire interdites.

6.1.4 Conclusion

Les différentes expérimentations que nous avons menées avaient pour but d'évaluer le modèle d'agents mobiles coopérants, présenté dans le chapitre 5. Pour cela, nous avons réalisé un service de localisation dans des environnements doublement mobiles, matériel et logiciel, avec des configurations à large échelle. Nous avons vu dans le chapitre 4 que, pour pallier au problème principal de reconfigurations constantes de ce type d'environnement, tant au niveau matériel que logiciel, nous proposons une nouvelle méthode de conception des applications réparties. Celle-ci repose sur un système autorisant uniquement des communications locales entre paires d'agents, *i.e* les coopérations. La principale difficulté est donc de localiser et de mettre en relation les partenaires d'une coopération. Notre méthode se base sur l'utilisation de deux grands types d'agents, les lourds et les légers, ainsi que la mise en place d'une couche d'ambiance, présentée dans le chapitre 3. Cette couche réalisée par les agents légers permet de cacher une partie du dynamisme de l'environnement aux agents lourds. Nous avons donc comparé cette approche avec deux autres méthodes afin étudier son efficacité.

Les différentes expérimentations réalisées viennent valider notre approche en montrant que l'utilisation de notre modèle permet d'augmenter l'efficacité des applications en diminuant le nombre de migrations nécessaires à la réalisation de leurs tâches (fig. 6.1). Cette augmentation se voit même accrue si le dynamisme physique est important (fig. 6.2). De plus, ce gain s'obtient en gardant une indépendance entre les agents (fig. 6.4) et leur autonomie par rapport à

la couche physique (fig. 6.3). Ceci représente deux critères fondamentaux pour résoudre les problèmes liés à la double mobilité et au passage à large échelle (cf 4.3)

Avec la mise en place de notre approche grâce à l'utilisation des deux grands types d'agents, nous obtenons une couche d'ambiance, réalisée par les agents légers, qui offre la possibilité aux applications métiers, construites sur des agents lourds, de se soustraire à la gestion du dynamisme de l'environnement et ainsi d'optimiser leurs déplacements qui leur garantit une amélioration de performance.

Nous avions bien précisé lors de la description des deux grands types d'agents (cf section 3.3 p.71) qu'il ne s'agissait aucunement de donner des types figés mais simplement un référentiel basé sur des critères de classification permettant de faciliter la conception. Il existe en fait une continuité allant des agents légers jusqu'aux agents lourds pouvant être utilisée par les concepteurs lors de la définition de leurs agents. Ainsi selon leurs besoins, ils pourront maximiser le nombre de rencontres ou minimiser le nombre de migrations (cf fig 6.5).

Enfin, nous avons aussi expérimenté le dimensionnement de la couche d'ambiance afin de trouver le nombre idéal d'agents légers permettant d'obtenir le meilleur rendement possible pour la couche d'ambiance. Nous avons vu (cf fig 6.6 et 6.7) que ce nombre idéal n'existe pas, la couche devant s'adapter à l'environnement afin de garantir un rendement satisfaisant. Pour cela, elle doit ajuster le nombre d'agents légers qui la composent en fonction de la taille du réseau sous-jacent. Nous devons donc proposer des mécanismes permettant d'obtenir des informations globales dans des environnements fortement dynamiques et gérées de manière purement locale. C'est l'objet de la prochaine section.

6.2 Équilibrage de charge

Dans cette section, nous nous intéressons à la capacité des agents légers à collecter et rassembler, sur des environnements dynamiques, des informations distribuées portant sur l'état global du système. Pour cela nous allons utiliser une migration libre basée sur des déplacements aléatoires. Ceux-ci sont largement utilisés dans les protocoles de recherche et de maintenance des topologie des réseaux P2P [GMS04], dans la communication par groupe dans les réseaux *ad hoc* [DSW02] et pour le routage dans les réseaux sans fil [BDDN01]. À titre d'exemple, nous décrivons un algorithme se basant sur les agents légers pour évaluer la charge des nœuds d'un réseau et pour opérer un équilibrage de charge.

Cet exemple d'équilibrage de charge permet de mettre en pratique plusieurs propriétés de notre modèle pour le traitement de la double mobilité. Pour commencer les agents n'ont pas besoins de connaître les nœuds ni de construire eux-mêmes une représentation du réseau. Chaque agent est totalement indépendant, il débute sans connaître le nombre d'agents de l'environnement, sans connaître l'architecture du réseau et en réalisant uniquement des coopérations indirectes. Ensuite, en paramétrant correctement leurs propriétés, les agents mobiles s'adaptent efficacement aux changements du contexte global. Et finalement, l'indépendance des agents permet une meilleure tolérance aux fautes tout en permettant de réaliser des coopérations avec des partenaires rencontrés inopinément sur un site.

6.2.1 Le problème de l'équilibrage de charge

Description du problème

Le problème de l'équilibrage de charge se pose lorsque l'on considère une application composée de plusieurs éléments s'exécutant sur un réseau. Chaque élément peut-être actif, un processus, ou passif, un fichier. Toute création d'un nouvel élément peut-être interne, lors de la création d'un nouveau processus, ou externe, lors de la création d'un fichier. À tout instant, un nouvel élément peut-être créé sur un nœud du réseau. La responsabilité de l'algorithme d'équilibrage de charge est de répartir ces nouveaux éléments sur les nœuds accessibles avec une faible charge.

Nous nous intéressons donc au cas de la répartition de charge qui est dit continue et dynamique dans lequel des éléments peuvent être générés à n'importe moment [SHK95]. Il est évident que les éléments peuvent tout autant disparaître à n'importe quel moment. Il s'agit généralement de la fin d'un processus ou de la destruction d'un fichier. Ceci diminuant la charge du site qui possédait l'élément disparu. Dans ce cas, la tâche de l'algorithme d'équilibrage de charge est de garantir que des éléments sont déplacés depuis un ou plusieurs nœuds surchargés vers ceux moins chargés.

Ce problème s'applique généralement dans des environnements où l'architecture du réseau est la plus stable possible et où les nœuds sont tous identiques, comme dans les cas d'un cluster par exemple. Dans ce contexte, lors de l'équilibrage les éléments sont déplacés sans qu'ils s'en rendent compte et sans qu'ils en aient fait la demande. Nous appliquons ce principe à une architecture plus dynamique, comme les grilles de calcul, où tous les éléments sont représentés par des agents qui acceptent d'être déplacés par le gestionnaire d'équilibrage. Dans ce cas, les sites peuvent eux aussi apparaître et disparaître au cours de l'exécution de l'équilibrage. Rappelons que nous partons de l'hypothèse que tous les nœuds ne sont jamais indéfiniment déconnectés.

Algorithme de base

La répartition de charge est réalisée à partir d'un service d'équilibrage composé d'au moins un agent mais plus généralement d'un ensemble d'agents s'exécutant en parallèle. Ceux-ci parcourent le réseau aléatoirement, allant de site en site, et pour chaque nœud visité, ils décident de déplacer les éléments en trop sur les nœuds dont la charge est trop élevée. Cette décision est basée sur la capacité des agents à percevoir la charge moyenne de l'ensemble des nœuds du réseau.

La moyenne globale du réseau est définie comme le nombre total d'éléments divisé par le nombre total de sites présents dans le réseau dynamique. Il est évident qu'il est impossible pour un agent de connaître avec exactitude ces nombres, c'est pour cela qu'il devra calculer approximativement cette moyenne de charge. Cette approximation est construite comme suit :

- Chaque agent gère une variable `nbSauts` qui est incrémentée de un à chaque fois que l'agent se déplace d'un site vers un autre. Notons que `nbSauts` *n'est pas* le compteur des différents sites rencontrés par l'agent, mais bien le nombre de migrations totales réalisées, incluant les déplacements vers des sites déjà visités.
- Chaque agent utilise aussi une variable `moyenne` pour compter le nombre d'éléments semblant être présents sur le réseau. À chaque fois

qu'un agent arrive sur un site, `moyenne` est (ré)évaluée en fonction du nombre d'éléments présents sur le nœud.

Les agents déterminent approximativement la moyenne de charge en calculant une nouvelle `moyenne` à partir de son approximation courante, du nombre d'éléments présents sur le nœud courant (`où.ComptElem`) et pondérée par le nombre de voisins (`où.NbVoisins`) :

$$moyenne = \frac{(moyenne * nbSauts * où.NbVoisins) + où.ComptElem}{(nbSauts * où.NbVoisins) + 1} .$$

Un agent commence ses déplacements aléatoires durant un minimum de `P` migrations avant d'utiliser son approximation de la variable `moyenne` pour équilibrer la charge. Après quoi, sur chaque site visité, il calcule l'approximation courante de `moyenne`. Ensuite, si le nombre d'éléments présents localement sur le site est plus grand que `moyenne`, l'agent régulateur en visite envoie un ordre de migration à certain éléments. Ces éléments doivent alors quitter le site en cours et migrer vers un site voisin. Dans notre cas, le choix du site se fait au hasard. Nous décrivons le comportement d'un agent régulateur de charge par la définition suivante :

Définition 6.1 (Comportement d'un Rg-agent) *Nous définissons le comportement de base d'un Rg-agent A visitant un site `où` par l'algorithme 6.5. La variable `moyenne` fait partie du contexte de l'agent (`A.moyenne`). La variable `ComptElem` du site en cours de visite (`où.ComptElem`) contient le nombre d'éléments présents localement.*

```
quand arrive(Agent A) sur où {
  A.moyenne = (A.moyenne*A.nbSauts*où.NbVoisins) + où.ComptElem / (A.nbSauts*où.NbVoisins) + 1;
  A.nbSauts = A.nbSauts + 1;
  if (A.nbSaut > P) {
    -- Fin du parcours initial
    int d = où.ComptElem − arrondi(A.moyenne);
    if (d > 0) { -- site surchargé
      ordonner la migration de d éléments locaux
    }
  }
}
```

Algorithme 6.5 - Comportement d'un Rg-agent

Propriétés de l'algorithme

Dans sa forme basique, l'algorithme 6.5 que nous venons de décrire comporte plusieurs propriétés intéressantes :

Propriété d'équilibrage de charge Si le nœud du réseau est surchargé, *i.e* il contient plus d'éléments que la charge moyenne actuelle, une des deux

choses suivantes peut arriver : Soit la charge moyenne actuelle augmente suffisamment pour que le nœud en question ne soit plus considéré comme surchargé, soit un ou plusieurs éléments quittent le site, que ce soit par une terminaison ou suite à un ordre de migration donné par un Rg-agent.
Il est intéressant de noter qu'à la différence des autres algorithmes de répartition [BFM98], notre approche ne requiert pas de connaissance globale de l'environnement. En particulier, aucun agent ne connaît le nombre de sites composant le réseau, ni précisément le nombre d'éléments présents dans l'environnement.

Propriété de convergence Si le système atteint un point fixe, *i.e* un point où plus aucun élément nouveau n'est créé, l'algorithme comporte une propriété forte : il atteindra un état où aucun site ne contient plus d'éléments que la moyenne de charge[10]. Cette propriété de convergence [Dij74] n'a pas besoin que le réseau sous-jacent devienne statique, les nœuds restent mobiles et peuvent rester complètement déconnectés durant un laps de temps fini.

Propriété de tolérance aux fautes L'algorithme est résistant aux fautes par nature[Lyu95]. La perte d'un Rg-agent peut être résolue en utilisant un ensemble d'agents. Si un agent est perdu ou bloqué sur son site (déconnecté) courant, les autres continuent la tâche de répartition de charge. Tant qu'au moins un agent survit, l'algorithme continue de fonctionner, quoiqu'avec une évidente perte de performance.

Nous avons utilisé le simulateur, décrit dans l'annexe A, pour évaluer expérimentalement ces différentes propriétés, en particulier les deux premières. De plus, les propriétés d'équilibrage de charge et de convergence ont été vérifiées sur un réseau de taille réduite en utilisant une description formelle écrite en TLA+ [Lam02].

6.2.2 Réactivité et coopération

L'algorithme de base que nous venons de présenter pour l'équilibrage de charge peut avoir une bien meilleure efficacité en effectuant plusieurs raffinages. Dans cette section nous présentons deux de ces raffinages possibles. Il s'agit d'un ajustement réactif et de l'utilisation du mécanisme de coopération directe.

Ajustement réactif

Si les agents s'occupant de l'équilibrage de charge calculent toujours leur approximation de la charge moyenne en partant de leur état initial, une longue exécution va induire de lentes mises à jour de cette moyenne. Cela vient de l'actuelle méthode de perception de la variation du nombre d'éléments de l'environnement. En d'autres termes, la méthode actuelle de perception devient de moins en moins pertinente pour permettre aux agents régulateurs de constater les variations du nombre global d'éléments. Ce problème vient du fait que les nouvelles visites

[10] Dans le cas où la moyenne n'est pas un nombre entier, l'état stable est atteint quand aucun site ne contient plus d'éléments que l'arrondi supérieur de la moyenne de charge. Notons que pour avoir cette propriété, la fonction d'arrondi utilisée dans la description doit arrondir au nombre supérieur.

de site ont un impact numérique de moins en moins important sur le calcul de la variable *moyenne*.

Cette situation peut-être problématique si le nombre global d'éléments de l'environnement vient à changer brusquement. Par exemple, imaginons une situation où le nombre total d'éléments est resté relativement stable pendant une longue période et que l'environnement enregistre de brusques, rapides et nombreuses créations d'éléments. Dans la version de base de l'algorithme, les agents régulateurs s'exécutant depuis un certain temps vont être particulièrement lents à déplacer les éléments des nœuds surchargés. Cet inconvénient peut être éliminé en laissant les agents percevoir l'instabilité de l'environnement grâce à l'utilisation d'un calcul d'écart type. Nous définissons l'écart type de la manière suivante :

$$\mathbf{et} = \sqrt{\frac{\sum_{i=1}^{i=N} (\mathbf{ComptElem}_i - M)^2}{N}}$$

où M est la moyenne de charge actuelle et N est le nombre actuel de sites (connectés) composant le réseau.

Les agents ne peuvent cependant pas calculer exactement cet écart type car ils ne peuvent ni récupérer la charge moyenne globale, ni connaître le nombre total de nœuds. Par contre, ils peuvent utiliser la même approche que celle servant à calculer la charge moyenne et obtenir des approximations successives de l'écart type. On obtient alors l'écart type décrit par la définition suivante :

Définition 6.2 (Écart type approché) *Nous définissons l'écart type « approché » à partir de la formule suivante, où* `moyenne` *est la valeur courant de la moyenne de charge calculée par l'agent et d est une taille de chemins prédéfinie.*

$$\mathbf{et}' = \sqrt{\frac{\sum_{i=1}^{i=d} (\mathbf{ComptElem}_i - \mathbf{moyenne})^2}{N}}$$

Le comportement d'un Rg-agent est donc le suivant : Si la différence entre deux valeurs successives de l'approximation de l'écart type $\mathbf{et}'$ est plus grande qu'une limite choisie, l'agent réinitialise sa variable courante *nbSauts* à une faible valeur. Cette mise à jour permet d'augmenter la réactivité de l'algorithme en donnant à une sous-séquence de visites plus de poids dans le calcul de l'approximation de la charge *moyenne*.

Coopération

Nous avons vu lors de la présentation de notre modèle d'agents mobiles coopérants (cf chapitre 5), que les agents peuvent évoluer indépendamment les uns des autres tout en engageant des phases de dialogue avec les partenaires qu'ils rencontrent à l'improviste au cours de leurs déplacements. Dans l'équilibrage de charge que nous étudions ici, nous pouvons mettre à profit la coopération pour affiner, à chaque étape de migration, l'approximation de la charge moyenne avec celles transportées par les partenaires présents et rencontrés sur les sites visités.

Avant de regarder en détails la manière de raffiner l'algorithme d'équilibrage en intégrant la coopération, rappelons sommairement son mécanisme de fonctionnement. Celui-ci se base sur une méthode de type Client/serveur et utilise le service d'annuaire local. Quand un agent

Paramètres du réseau	
Nombre de nœuds	50
Moyenne de voisins par nœud	4
Paramètres des agents	
Durée moyenne de transfert	50 μs
Durée moyenne de visite	0.5 sec.

TAB. 6.2 – Paramètres de simulation

commence une visite sur un site, il s'enregistre auprès du service d'annuaire et il entre dans l'état de coopération, comme le décrit l'automate de la figure 5.2 de la page 96. Dans cet état, l'agent peut initialiser (et/ou accepter) de nouvelles coopérations avec les partenaires locaux et utiliser (et/ou exécuter) les méthodes servant à la coopération. À tout instant, l'agent peut décider de ne plus accepter de demande et refuser toute nouvelle coopération. Cependant, il doit attendre que toutes les coopérations engagées soit terminées avant de sortir de l'état coopérant et d'entamer un nouveau déplacement.

L'intégration de la coopération peut permettre d'augmenter l'efficacité de l'algorithme de régulation. En effet, lorsque qu'un agent régulateur visite un site, il envoie une requête au service d'annuaire local afin de savoir s'il y a sur le site d'autres agents participant à la fonction de répartition de charge. Si tel est le cas, il entame une phase de coopération avec chaque agent trouvé par le service d'annuaire. Cette coopération consiste à récupérer l'état courant des autres agents et de mettre à jour l'approximation de la charge moyenne avec celle des autres agents. Pour cela, l'agent régulateur recalcule sa variable *moyenne* de la façon suivante :

$$\mathbf{moyenne} = \frac{\sum_{i=0}^{i=p}(\mathbf{moyenne}_i * \mathbf{nbSauts}_i)}{\sum_{i=0}^{i=p}\mathbf{nbSauts}_i}$$

Où p est le nombre d'agents coopérant trouvés sur le site en cours, incluant l'agent initiant la coopération.

La mise en place de cette phase de coopération permet de rassembler les résultats partiels obtenus par chaque agent présent sur le site lors de leurs déplacements aléatoires réalisés sur des parties différentes du réseau. La réunion de ces résultats permet d'affiner l'approximation de la charge moyenne de l'ensemble du réseau.

6.2.3 Évaluation des performances

Pour évaluer notre algorithme d'équilibrage de charge, nous avons utilisé notre simulateur en démarrant d'un état initial de l'environnement où les éléments sont répartis de manière inégale. Ils sont concentrés sur une moitié du réseau afin de créer dès le départ un déséquilibre de la charge. Nous avons défini un ensemble de paramètres communs à toutes les simulations que nous exposons dans le tableau 6.2.

La figure 6.8 montre la première simulation que nous avons réalisée. Il s'agit ici d'évaluer l'impact du dynamisme physique, *i.e* la fréquence de déplacement des nœuds, sur les performances de notre algorithme de répartition en fonction du nombre d'agents régulateurs utilisés. Pour cela, nous avons créé un environnement stable sur la taille du réseau (50 sites) et sur le

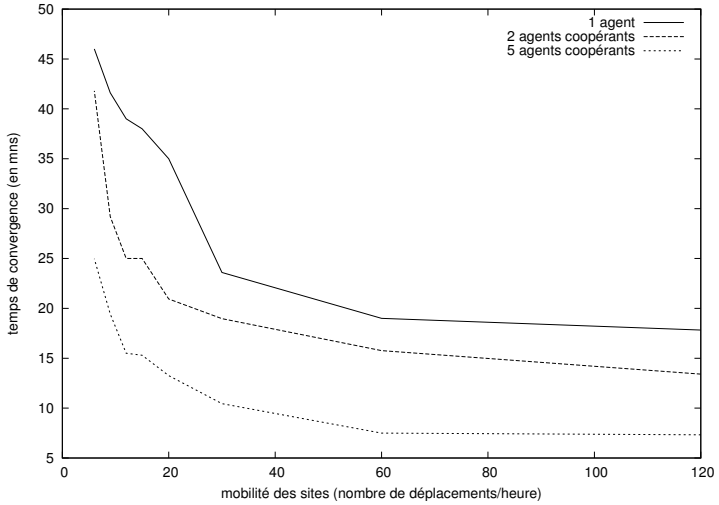


FIG. 6.8 – Impact de la mobilité des nœuds

nombre d'éléments (150 entités). La mobilité des sites est spécifiée par un nombre moyen de déplacements effectués en une heure. Pour chaque paramètre nous avons réalisé une vingtaine de simulations. Les résultats recueillis montrent qu'une grande mobilité des sites permet de diminuer la durée nécessaire pour obtenir un système équilibré et donc d'obtenir une augmentation de l'efficacité de l'algorithme. De plus, nous pouvons noter qu'à partir d'un certain dynamisme, l'augmentation de la mobilité des sites influence très faiblement les résultats. Dans nos simulations, nous le constatons dès lors que les sites se déplacent en moyenne une fois par minute (60 déplacements par seconde). Notons que les agents se déplacent 100 fois plus rapidement que les sites.

Dans les expérimentations suivantes, représenté par les figures 6.9, 6.10 et 6.11, nous avons simuler des brusques montées en charge du système afin de tester l'effet des différents raffinages que nous avons proposés dans la section précédente. Pour cela, nous évaluons le temps de réponse nécessaire aux agents régulateurs pour atteindre un système équilibré, *i.e* dans notre cas un système où l'écart type est égal à zéro.

Pour commencer, nous avons comparé l'efficacité du mécanisme de réactivité avec l'algorithme de base. Pour cela, nous avons évalué le temps nécessaire à 5 agents coopérants pour équilibrer le système. Pour simuler la montée en charge, nous avons effectué 5 brusques séries de créations de 40 éléments, successivement espacées de 3 minutes. Les résultats sont présentés sur la figure 6.9. Nous pouvons constater que non seulement les montées en charge sont plus rapidement absorbées avec le mécanisme de réactivité, mais qu'en plus l'algorithme de base met largement plus de temps à équilibrer le système. Nous pouvons donc voir que le mécanisme de réactivité permet d'accroître largement l'efficacité de l'algorithme de base

Ensuite nous avons évalué l'impact de la coopération sur la réactivité des agents. Pour cela nous avons utilisé la même approche en créant successivement quatre séries de brusques créations de 50 éléments. Les séries étant provoquées de manière périodique. Nous avons donc évalué l'efficacité apportée par la coopération en mettant en comparaison le temps d'équilibrage nécessaire à un agent et celui de 5 agents coopérants. Les résultats de la figure 6.10 viennent confirmer l'importance de la coopération. En effet, nous pouvons constater que la diminution

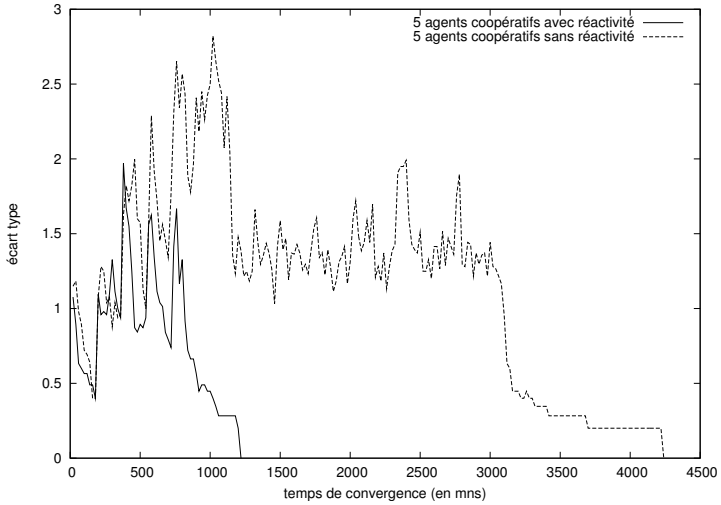


FIG. 6.9 – Impact de la réactivité

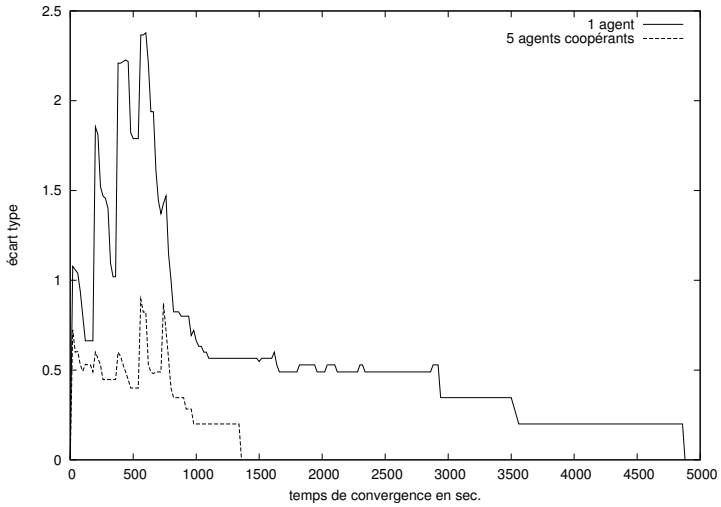


FIG. 6.10 – Impact du nombre d'agents

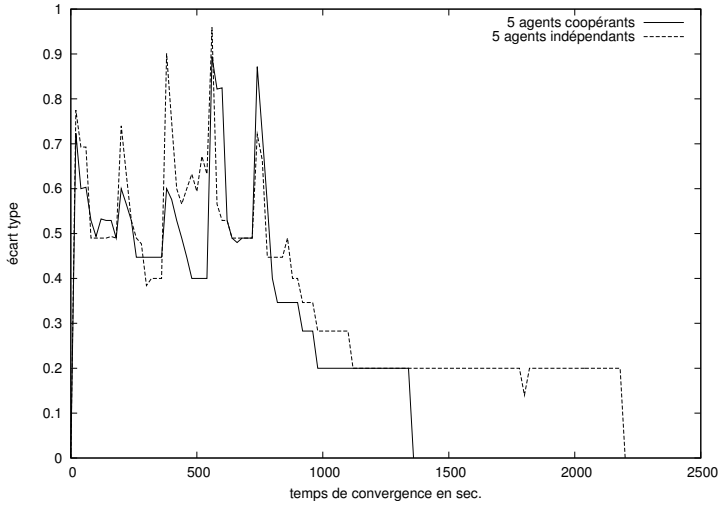


FIG. 6.11 – Impact de la coopération

du temps d'équilibrage est presque d'un facteur 5 et, de plus, que la réactivité est nettement meilleure avec la coopération qui absorbe beaucoup plus vite les montées en charge.

Enfin pour bien confirmer que l'augmentation de l'efficacité de l'équilibrage est dû à la coopération, nous avons réalisé une dernière simulation en comparant un équilibrage réalisé par des agents indépendants avec une régulation réalisée par des agents coopérants. Pour cela, nous avons repris la même configuration que la simulation précédente et nous avons fixé, dans les deux cas, le nombre d'agents régulateurs à 5. La figure 6.11 montre que les agents indépendants vont avoir un plus grand écart type que les coopérants et que le temps de réponse nécessaire à l'équilibrage est là aussi plus faible lors de l'utilisation de la coopération.

6.2.4 Conclusion

À travers l'exemple de l'équilibrage de charge, nous venons de présenter un méthode basée sur notre modèle d'agents mobiles coopérant permettant d'obtenir un algorithme fiable et réactif pour les environnements fortement dynamiques. Pour cela, nous nous reposons sur des agents qui sont totalement indépendants les uns des autres, qui parcourent aléatoirement l'environnement et qui peuvent engager des coopérations s'ils rencontrent inopinément des partenaires. Cet algorithme possède des propriétés intéressantes dont les deux principales sont la continuité de la régulation malgré la double mobilité et l'existence d'un point fixe si le nombre d'éléments se stabilise.

Nous pouvons mettre en avant que l'utilisation de notre modèle permet aux agents d'être démarrés sans aucune connaissance de leur environnement, en particulier en ce qui concerne le support physique. Ceci donne à notre approche des similitudes avec les algorithme autostabilisants. En effet, les expérimentations montrent qu'un ensemble non-vide d'agents, ne possédant aucune information initiale sur le contexte, est capable de calculer une information globale tels que la moyenne de charge du réseau et l'écart type.

Cette manière de calculer une information globale peut facilement s'étendre à d'autres paramètres. Par exemple, en marquant un site et en calculant le nombre moyen de migrations réalisées pour revenir sur ce site, un agent peut estimer la taille d'un réseau possédant une

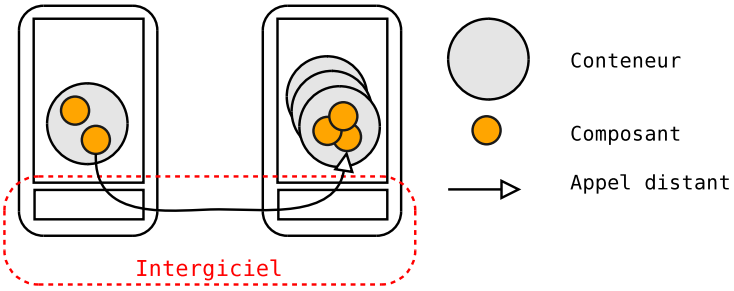


FIG. 6.12 – Composant classique

architecture dynamique[11]. En récupérant différentes sortes d'informations sur les sites du réseau, les agents peuvent aussi réguler plus précisément l'activité du réseau, par exemple en approximant la quantité globale de ressources disponibles (puissance processeur, espace de stockage, etc.). Avec de telles approximations, les agents régulateurs peuvent même être utilisés pour réaliser une stratégie de régulation à plus long terme : en fonction des ressources globales disponibles, ils pourront accorder ou différer de nouvelles créations d'éléments.

Nous avons vu dans la section précédente, que la couche d'ambiance a besoin d'adapter sa configuration en fonction du nombre de sites composant le réseau. Cette couche est réalisée à l'aide d'un ensemble d'agents légers qui adoptent des déplacements aléatoires pour parcourir l'environnement. La taille de cet ensemble devant correspondre à 5% de celle du réseau. Nous pouvons donc utiliser notre méthode d'approximation d'informations globales pour estimer d'un côté le nombre de nœuds de l'environnement et d'un autre côté calculer le nombre d'agents légers composant la couche d'ambiance. Avec ces deux approximations, nous pouvons dynamiquement adapter la taille de la couche en créant ou détruisant des agents de localisation et donc rester proche des 5% visés.

6.3 Service de composants

Nous avons présenté le service de localisation qui permet de réaliser la couche d'ambiance et un mécanisme de calcul d'informations globales sur l'environnement permettant ainsi de garantir une certain stabilité de l'efficacité de la couche. Dans cette section nous présentons une première idée d'application métier reposant sur l'utilisation de la couche d'ambiance, il s'agit d'un service de composants mobiles.

Nous avons vu qu'avec la double mobilité, une conception classique s'appuyant sur le modèle client/serveur est particulièrement difficile à mettre en œuvre dans des environnements fortement dynamiques. Or, cette méthode est la plus largement utilisée dans les intergiciels à composant tel Weaver [WMC04] ou CCM [Mis99]. C'est pourquoi, nous proposons une notion de composant mobile qui vise à répondre à des besoins propres aux composants en terme de souplesse de déploiement et d'adaptation dans un tel contexte dynamique. Pour cette approche nous allons utiliser notre modèle d'agents mobiles coopérants en faisant jouer aux agents lourds le rôle de containers. La mobilité des composants est ainsi prise en charge par la mobilité des agents.

[11] Avec la charge moyenne, ils peuvent même estimer le nombre total d'éléments dans l'environnement.

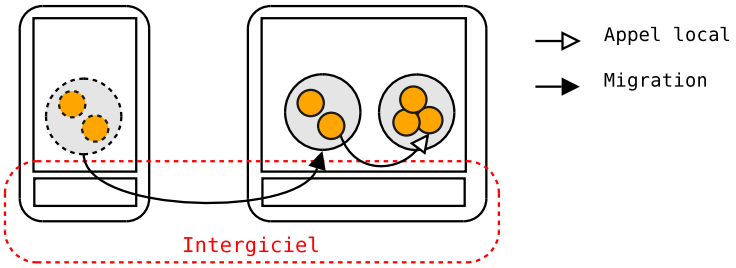


FIG. 6.13 – Composants Mobile

Nous commençons par un rappel des propriétés des composants logiciels où nous abordons en particulier le problème de la mobilité des composants. Nous proposons une architecture générale pour un tel service de composants mobiles et abordons le problème de leur localisation en corrélation avec l'utilisation de la couche d'ambiance. Pour terminer, nous illustrons l'usage des composants mobiles, sous une forme simulée, dans une application de calcul parallèle incluant la mise en œuvre d'une régulation de charge que nous avons déjà vu.

6.3.1 Architecture du modèle à composants mobiles

D'un point de vue architecture logicielle, une application est considérée comme une collection de composants logiciels indépendants interconnectés [MP00]. Un composant est vu comme une boîte noire accessible au travers d'une interface et de protocoles associés. Créer une application revient alors à assembler (connecter) différents composants et à définir leurs interactions [RM00].

Pour pouvoir mettre en place un composant, on utilise une plate-forme qui prend en charge l'interaction entre les composants ainsi que leur déploiement sur un ensemble de machines. Ces serveurs d'accueil gèrent plusieurs containers, chacun encapsulant un ensemble de composants. Pour des raisons d'efficacité et/ou de tolérance aux fautes, les containers peuvent être dupliqués. Un container gère l'état de chaque composant qu'il encapsule, intercepte les appels à ces composants et fait transiter les appels à distance par un bus logiciel. On peut voir ces différents éléments sur la figure 6.12.

Dans les architectures hybrides, nous sommes face à des contextes de déploiement très hétérogènes qui demandent de posséder des outils d'administration et de supervision particuliers [RB05]. Pour appliquer l'approche composant aux environnements dynamiques et pour en faciliter l'administration ainsi que la supervision, nous proposons de nous baser sur notre modèle d'agents mobiles coopérants pour proposer une couche de service de composants mobiles. Une telle approche consiste à faire jouer à des agents mobiles le rôle de containers de composants. Ces agents containers mobiles (ACM) ont pour tâche d'intercepter les appels inter-composants. Lorsqu'un composant C_1 d'un container ACM_1 appelle un composant C_2 encapsulé dans un autre container ACM_2, l'agent container ACM_1 doit alors identifier le composant cible (C_2), le localiser et se déplacer vers un(le) site sur lequel il existe un agent container (ACM_2) qui gère le composant appelé. Les deux containers coopèrent alors localement afin de réaliser l'appel de méthode entre les deux composants. La figure 6.13 montre ce mécanisme.

Intérêt de la mobilité des composants

En utilisant les agents mobile pour représenter les containers de composants, nous pouvons obtenir les propriétés suivantes :

Contrôle de l'accès à un composant Le composant n'est accessible que sur les sites qu'il visite. C'est à la fois une restriction par rapport au modèle classique de type RPC qui permet un accès de n'importe où[12], mais c'est aussi un moyen d'assurer des restrictions d'accès pour des raisons de sécurité : les agents n'iront pas sur des sites peu sûrs et inversement, un site peut se protéger des agents issus de sites suspects.

Adaptabilité Un des problèmes majeurs des plates-formes à composants est celui de l'adaptation : un composant doit pouvoir être adapté dynamiquement pour répondre à diverses contraintes de caractère non fonctionnel (sécurité, qualité de service)[BCL+04] mais aussi aux besoins de ses usagers.

La mobilité des composants facilite et enrichit les possibilités de déploiement et de reconfiguration dynamique : on peut spécifier diverses contraintes concernant le domaine sur lequel le composant doit être déployé, le nombre de containers le gérant, éventuellement le schéma de migration (au hasard, selon un chemin, à la demande d'autres agents containers). Le déploiement du composant peut aussi évoluer dynamiquement en fonction des besoins, en fonction de la charge, soit pour augmenter le parallélisme durant l'exécution, soit pour favoriser les interactions locales par un placement optimal.

Coopération des agents containers Un aspect intéressant de cette approche est que les agents containers, en gérant les composants qu'ils contiennent, peuvent coopérer pour s'adapter aux contraintes logicielles et/ou physiques. On peut envisager que deux agents containers échangent leurs composants pour permettre, par exemple, la migration plus rapide d'un des deux agents : autrement dit, l'agent container se fait remplacer.

Conception du service

L'implantation des composants mobiles s'appuie donc sur notre modèle d'agents mobiles coopérants (AMC) en faisant jouer aux agents le rôle de containers. Ces AMC encapsulent un ensemble de composants et autorisent uniquement les interactions locales, lorsque les AMC sont présents sur un même site

Comme pour les composants classiques, le déploiement et la localisation des composants mobiles sont réalisés à partir de référentiels d'interface (interface repositories) et d'implantations (implémentation repositories) à la CORBA [GGM99]. La figure 6.14 illustre cette approche. Une base d'interfaces de composants mobiles recense les composants connus. Un composant mobile actif qui recherche un service peut consulter une telle base pour demander le déploiement du composant recherché ou demander à un service de localisation un domaine de sites sur lesquels il est susceptible de trouver ce type de composant.

Le déploiement d'un composant mobile fait appel à une base d'implantations de composants et à un paramétrage initial. Les paramètres spécifient le nombre initial d'instances, le

[12] À ceci près qu'il faut pouvoir alors communiquer entre le serveur et le client (sans déconnexion).

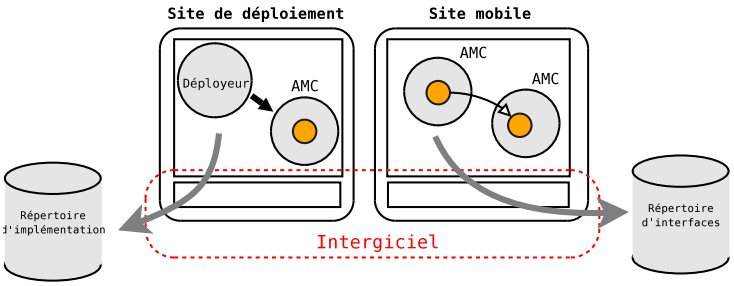


FIG. 6.14 – Déploiement et localisation des composant

type de mobilité (réactive ou proactive) des ACM's supports, les contraintes sur la mobilité (restreinte à un sous-domaine ou un type de sites), etc. Des sites dédiés peuvent contenir de telles bases et le déploiement se fait alors à partir de ces sites.

6.3.2 Utilisation de la couche d'ambiance

Nous venons de voir, que pour permettre l'utilisation du modèle de composants sur les environnements dynamiques, nous faisons réaliser la fonction de container à des agents mobiles. Lors d'un appel de méthode par un composant, le container demande à la base d'interface si le composant visé existe ou s'il faut le déployer. Si le composant cible existe déjà et qu'il est géré par un autre container en cours d'évolution dans le contexte, l'AMC du demandeur devra alors faire appel à la couche d'ambiance pour localiser le container de la cible et ainsi migrer vers lui.

Si on regarde les AMC d'un peu plus près, nous pouvons constater qu'ils possèdent toutes les caractéristiques des agents lourds, l'un des deux grands types d'agents que nous avons déjà décrits (cf 3.3 p. 71). Pour commencer, les composants réalisent généralement des fonctions métiers qui vont nécessiter de longues phases de calcul local et donc limiter la fréquente de déplacement des containers. Ensuite, la gestion simultanée de plusieurs composants rend les AMC volumineux et va leur imposer de longs temps de transmission. Nous avions vu que toutes ces caractéristiques ne permettent pas aux agents lourds, et donc aux AMC, de s'adapter facilement au dynamisme de l'environnement. De plus, la fonction propre de container doit être la plus efficace possible et par conséquent ne doit pas être gênée par des fonctions relevant du système.

C'est pour ces différentes raisons que les AMC vont utiliser la couche d'ambiance afin d'avoir une représentation plus stable de leur environnement en déléguant la gestion du dynamisme aux agents légers et en se focalisant sur la fonction de container. Ainsi en fonction des appels effectués par leurs composants internes, les AMC interrogent le service de localisation pour trouver les containers cibles et peuvent choisir au mieux l'itinéraire à suivre.

6.3.3 Mise en place de la régulation

La gestion des composants constitue, en plus de l'utilisation de la couche d'ambiance, un champ d'application pour la méthode de répartition de charges que nous avons vu dans la section précédente. En effet, généralement, les composants cherchent uniquement à réaliser leur tâche sans se soucier du site les exécutant. Ils acceptent totalement la répartition du moment

qu'ils peuvent continuer leur calcul sans interruption notable. Pour réaliser cette répartition, nous distinguons deux composants : un *composant applicatif* de calcul parallèle et un *composant régulateur* qui contrôle dynamiquement le déploiement du composant de calcul.

Le déploiement du composant applicatif se fait en créant un agent container initial qui commence le calcul. Celui-ci crée des agents containers qui, eux-mêmes, peuvent en créer d'autres (calcul parallèle diffusant). Ces agents containers ne migrent pas d'eux-mêmes mais acceptent de se déplacer à la demande. Le déploiement du composant régulateur se fait en créant un ou plusieurs agents containers exécutant la régulation. Ces agents migrent au hasard. Les instances de composants encapsulés dans les containers coopèrent selon le schéma classique client-serveur, lorsque deux agents containers régulateurs se rencontrent. Ils interagissent aussi avec les agents containers du composant de calcul en émettant vers ceux-ci des requêtes de migration.

Dans une telle application, le déploiement du composant de calcul peut se faire sur le site dédié. Le container initial peut alors déclencher la création d'autres containers pour paralléliser le calcul. Le composant régulateur est créé sous la forme d'un nombre fixé d'agents containers (au moins un) en lui donnant l'identité du composant mobile de calcul qu'il doit superviser. Grâce à un service de base offert par l'intergiciel de gestion des agents, un agent container connaît le nombre d'agents containers de calcul présents sur un site qu'il visite. Par une visite continue et aléatoire des sites, il totalise le nombre d'agents containers de calcul rencontrés et en déduit une approximation de la moyenne par site. Grâce à cette moyenne, tout container régulateur est alors en mesure de décider si des containers de calcul doivent être déplacés. Ces requêtes de migration se font par interaction locale entre le container régulateur et les containers de calculs présents sur le site.

6.3.4 Conclusion

Nous venons donc de présenter une première application métier qui utilise la couche d'ambiance. Il s'agit de transposer le modèle de programmation par composants dans les environnements dynamiques en définissant les containers comme des agents mobiles (AMC). Leur rôle est alors de mettre en relation locale les composants afin de limiter les effets du dynamisme, en particulier les ruptures de connexion lors de la communication à distance.

Pour permettre le rapprochement des composants, les AMC doivent être capables d'identifier et de localiser les composants œuvrant dans l'environnement. Pour cela, ils utilisent la couche d'ambiance qui réalise pour eux la fonction de localisation. Ceci permet aux AMC de déléguer une fonction système à la couche d'ambiance et de se concentrer sur leur fonction de container. La couche d'ambiance permettant de garantir un nombre de migrations moyen pour atteindre un élément cible, les AMC peuvent évaluer le meilleur chemin à suivre en fonction des demandes émanant de leurs composants internes.

Pour permettre d'améliorer l'efficacité des containers et faciliter la gestion des composants, nous avons aussi proposé de mettre en place le mécanisme de régulation à partir de l'approximation d'information globale. Ainsi, les containers pourront se déplacer pour équilibrer la charge d'un site, mais aussi faire des échanges de composants afin d'équilibrer la gestion interne des AMC.

6.4 Agenda

La dernière partie de se chapitre porte sur une proposition d'application orientée client se basant sur l'utilisation de la couche d'ambiance. Pour cela, nous allons utiliser notre modèle d'agents mobile coopérants pour décrire une application distribuée qui permet de gérer des informations personnelles (PIM) et d'entreprises et qui évolue dans un environnement dynamique.

Ce genre d'application, type Outlook ou Evolution, permet de partager les données personnelles de l'ensemble des membres d'une entreprise. On nomme généralement par PIM les informations concernant les carnets d'adresses, les agendas ou encore les e-mails. Ces applications possèdent habituellement un ensemble de fonctionnalités d'aide à la gestion de ces données. La fonction qui nous intéresse ici est la planification de réunion à partir de partenaires identifiés appartenant au carnet d'adresses de l'entreprise. Cette planification suit un scénario standard que nous souhaitons modifier à l'aide de notre modèle.

6.4.1 L'application classique

Dans la quasi totalité des cas, les applications PIM sont fondées sur un modèle fortement centralisé où les informations sont stockées sur un serveur de référence. Les utilisateurs devant alors s'y connecter pour déposer et/ou consulter leurs informations personnelles ou celles de leurs collègues. Dans ce contexte, le scénario classique pour la réalisation d'une réunion est le suivant :

1 - l'utilisateur se connecte au serveur central afin de récupérer les données de l'ensemble des membres concernés[13]

2 - puis il consulte manuellement les différents agendas afin de trouver le rendez-vous le mieux adapté ;

3 - ensuite il envoie une proposition à toutes les personnes concernées afin qu'elles valident le rendez-vous.

4 - Enfin si tous les participants ont validés, la réunion est intégrée dans le serveur central. Dans le cas contraire l'utilisateur recommence le scénario depuis l'étape 1.

Ce scénario est parfaitement adapté aux architectures où les utilisateurs utilisent uniquement des unité fixes pour la gestion de leurs agendas. Cependant l'émergence des unités mobiles, comme les PDA ou autres Smartphones, permet aux utilisateurs de transporter leurs agendas aux cours de leurs différents déplacements. Cette possibilité va remettre en cause les applications utilisant le scénario classique ci-dessus..

6.4.2 Utilisation de la mobilité

La première solution envisagée, pour offrir la mobilité aux utilisateurs, est de maintenir en place cet algorithme en imposant des synchronisations avec le serveur central assez fréquentes pour maintenir une cohérence minimale. Cependant, ces synchronisations demandent une utilisation régulière et volumineuse de la bande passante. Or cela ne peut s'associer avec

[13] une amélioration possible étant de récupérer les données semaine après semaine [GOP02]

des réseaux asynchrones et à faible débit comme ceux utilisés avec les unités mobiles, comme nous l'avons décrit dans la partie 1.1.3.

Pour apporter une solution à la prise en charge de la mobilité des utilisateurs, nous devons repenser la manière d'envisager les applications de gestion de PIM. Dans le modèle centralisé, les données de références sont stockées sur le serveur central et les unités en déplacement possèdent une copie plus ou moins à jour de ces données de référence. Lorsqu'une modification est apportée sur un des deux supports, une synchronisation est opérée et l'utilisateur doit géré lui-même les conflits. Pour mieux prendre en compte la mobilité nous allons inverser cette méthode de gestion des données personnelles. Ainsi, les données de référence sont à présent celles véhiculées avec les unités mobiles et celles stockées sur le serveur de référence servent uniquement de sauvegarde. Ceci correspond donc parfaitement à notre modèle qui considère que les serveurs ne représentent qu'un service d'exécution et que les données sont dans les agents mobiles. Ici, chaque agenda sera représenté par un agent se déplaçant avec son propriétaire, il pourra rester sur l'unité mobile lorsqu'elle est déplacée par l'utilisateur ou bien être sur des machines différentes dans le cas où l'utilisateur ouvre des sessions différentes sur des ordinateurs différents (domicile, travail, client ect.)..

Avec cette nouvelle référence qui est en mouvement, la création de rendez-vous va demander d'aller chercher l'agenda des participants sur leur unité en déplacement. Pour cela, nous proposons d'utiliser notre modèle d'agents mobiles qui permettra de prendre en compte la mobilité par l'utilisation de la couche d'ambiance. Nous proposons donc un autre scénario pour la planification de rendez-vous coopératif à l'aide d'agents mobiles :

1 - l'utilisateur informe l'application d'un ensemble de membres participants à la réunion et de règles pour son organisation, comme un intervalle de temps ;

2 - l'application crée un agent mobile avec l'ensemble des membres de la réunion ;

3 - l'agent va chercher chacun des agendas concernés et va mettre en corrélation les informations recueillies afin de trouver plusieurs solutions pour la réunion ;

4 - l'agent retourne vers son créateur et lui soumet l'ensemble des propositions calculées ;

5 - l'utilisateur va ordonner les propositions et l'application va renvoyer l'agent. Il pourra aussi éliminer certaines solutions si nécessaire

6 - l'agent propose à chaque participant l'ordonnancement du créateur afin qu'il valide ses solutions préférées.

7 - l'agent revient à son créateur en lui indiquant toutes les validations des participants.

8 - Le créateur choisit alors la date finale, il valide la réunion et envoie l'agent informer tous les participants de la date choisie.

Durant l'exécution de l'application, les agents vont se déplacer pour trouver les agendas désirés et seront amenés à rencontrer des homologues en cours d'exécution sur les sites. Cette possibilité doit être mise à profit afin d'aider les agents dans leur tâche. Pour ce faire, nous présentons maintenant un peu plus en détail la manière dont l'agent va trouver et récupérer des informations concernant la tâche qu'il doit accomplir.

L'agent créé possède au démarrage un ensemble de cartes de visites électroniques au format VCARD [DH98] permettant de savoir quel est l'identifiant de l'agent représentant l'agenda du participant cible et quelle est sa localisation de référence, une sorte de port d'attache pour nos unités naviguantes. Nous prenons comme hypothèse que chaque participant possède un domaine originel, en d'autres termes que nous connaissons le minimum d'informations nécessaires permettant d'améliorer la recherche de l'agenda en déplacement, en particulier retrouver facilement la *trace de visite* si la couche d'ambiance ne peut la fournir directement.

6.4.3 Les coopérations utilisées

Pour l'organisation d'une réunion dans le cadre d'une application mobile, nous pouvons identifier deux grandes sous-tâches : la localisation des agendas et la constitution de l'ensemble des solutions envisageables. Nous allons mettre à profit la capacité de coopération des agents mobile pour améliorer ces deux tâches.

Pour la localisation des représentants des agendas, l'agent réunion coopère avec le service d'ambiance afin de trouver trace d'un des partenaires. Si la couche d'ambiance, peut fournir une politique de guidage, l'agent réunion va alors se déplacer vers le partenaire correspondant. Dans le cas contraire où aucune politique ne peut être fournie, l'agent réunion se déplace vers un site le rapprochant du domaine originel d'un des partenaires. Pour cela, nous mettons des agents fixes à l'entrée de ces domaines afin que la couche d'ambiance puisse fournir une destination par défaut. En fait, les agents réunions vont déléguer la tâche de localisation à la couche d'ambiance et vont se concentrer sur la constitution de l'ensemble des solutions possibles.

Pour créer cet ensemble, l'agent réunion est créé avec un intervalle de dates défini lors de son initialisation. Lorsque un agenda cible est trouvé, l'agent récupère les créneaux libres dans l'intervalle et les ajoute à l'ensemble des solutions déjà récupérées. En fait, l'agent met en corrélation les créneaux récupérés avec ceux qu'il transporte et va pouvoir éliminer les solutions non valides. Le but de l'agent réunion est donc de converger le plus rapidement possible vers un ensemble minimal de solutions. Pour cela, l'agent va prendre en compte les rencontres qu'il peut faire durant ses déplacements avec d'autres organisateurs de réunion. Il va donc coopérer directement avec eux pour échanger les informations véhiculées. Ici, il s'agit principalement de récupérer les disponibilités des contacts recherchés et ainsi éviter de transporter des informations inutiles.

Prenons l'exemple d'un agent qui cherche à organiser une réunion pour monsieur X avec un ensemble de contacts, comprenant monsieur Y, entre la semaine 23 et 24. Cet agent en rencontre un autre créé par monsieur Y qui transporte l'information que monsieur Y est occupé pendant la semaine 23. L'agent de monsieur X peut alors ignorer toutes les informations des autres contacts de la semaine 23. La coopération directe permet donc d'améliorer le calcul d'une date de réunion en réduisant le nombre de solutions envisageables, et par conséquent le temps des traitements locaux, mais aussi en diminuant le volume des informations transportées, permettant d'accélérer les temps de transfert et donc la mobilité des agents.

En fait, nous avons étendu le mécanisme de rumeurs, utilisé pour la couche d'ambiance, au niveau applicatif. En permettant les coopérations entre les agents « réunion », nous permettons aux rumeurs de se propager à l'ensemble de l'application de PIM. Notons que pour encore améliorer son efficacité, nous pouvons parfaitement envisager de faire des coopérations avec d'autres agents appartenant à des applications utilisant l'agenda distribué. Par exemple, si un participant X a planifié ses vacances pour un certain intervalle et qu'il crée un agent cher-

chant un voyage, lors de la rencontre entre l'agent réunion et l'agent voyage, le créneau des vacances pourra être éliminé pour l'organisation de la réunion. Nous pourrions aussi parfaitement étendre ce phénomène en intégrant à tout agent l'algorithme de surcharge pour le niveau applicatif. Ainsi tout agent de l'environnement serait vecteur d'informations pour les applications distribuées en cours d'exécution.

Conclusion

Dans cette thèse, nous nous sommes intéressés aux environnements dynamiques à grande échelle supportant la mobilité des sites. De tels environnements apparaissent à partir du moment où l'on étend Internet par des bornes d'accès (hotspot) et par la mise en place de sous-réseaux autonomes décentralisés (ad'hoc). Un utilisateur nomade peut alors se déplacer avec son unité mobile en changeant dynamiquement de sous-réseau autonome. Ces déplacements physiques, couplés aux morcelages des réseaux autonomes, apportent aux environnements une *mobilité matérielle* qui impose un fort dynamisme physique ne pouvant être pris en compte par les méthodes classiques de gestion de la répartiton.

Dans un tel contexte, nous avons tenté d'évaluer les propriétés propres des agents mobiles qui semblent s'adapter aux environnements où le système n'offre qu'une gestion locale du contexte. Par leur autonomie, leur adaptation et leur mobilité, les agents mobiles sont compatibles avec la mobilité des sites. En effet, la plupart des expérimentations faites montrent que les performances applicatives des agents mobiles, non seulement ne sont pas réduites par la mobilité des sites, mais sont plutôt bien conservées, voire légèrement améliorées. Les premiers résultats obtenus par des simulations confirment donc les potentialités des agents mobiles pour développer les applications réparties dans des contextes de communication très dynamiques.

Cependant l'utilisation des agents mobiles apporte un second niveau de mobilité, *i.e la mobilité logicielle*, qui pose un problème dès lors que l'on souhaite utiliser un service précis. Dans le cas où le système gère uniquement son contexte local, cette ***double mobilité*** pose le problème de la détermination de la localisation de l'agent applicatif, réalisant le service, ainsi que la détermination du chemin permettant de l'atteindre. Nous avons donc proposé un service de localisation dont l'objectif est d'offrir une représentation de l'environnement proche de la réalité. Ce service nous a permis de mettre en avant l'adéquation des agents mobiles coopérants avec les environnements dynamiques à grande échelle. Nous avons donc expérimenté la localisation des agents mobiles applicatifs grâce à un service réparti de nommage qui utilise la propagation de rumeurs. Nous avons donc pu mettre en évidence quelques propriétés intéressantes :

Tolérance aux fautes : la propagation des rumeurs par des agents dédiés fournit une solution tolérante aux fautes : la disparition d'un ou plusieurs agents peut être détectée et compensée par la création de nouveaux agents de façon simple. De plus, la disparition des sites n'affecte pas le service de localisation.

Performance : le nombre de migrations nécessaires pour qu'un agent trouve le service (agent) cible est largement diminué par rapport à la méthode de recherche classique. De plus, l'influence de la taille du réseau étant relativement limitée, les résultats obtenus peuvent être considérés comme satisfaisants.

Faible coût : lorsque l'on construit un service de localisation sur des agents dédiés, leur nombre peut rester relativement faible par rapport à la taille du réseau et au nombre de services à référencer. Cette propriété permet de minimiser les coûts de traitement du service de localisation et surtout permet de ne pas surcharger le réseau de communication qui possède une faible bande passante dans les environnements sans fil.

Adaptation : En fonction des besoins de recherche des agents, le service de localisation peut soit maximiser la rapidité de localisation du service recherché, soit minimiser le nombre de migrations nécessaires pour atteindre le service cible. Cette propriété d'adaptation permet de configurer dynamiquement le service de localisation.

Grâce à la réalisation du service de localisation, nous avons montré que les agents mobiles permettent d'obtenir un certain nombe de propriétés correspondant aux besoins des environnements dynamiques à large échelle. En utilisant ce service, nous avons aussi étudié différentes applications, comme un service de composants, qui permettent de transposer des méthodes de conception classique dans des environnements dynamiques et, ce, avec un effort d'adaptation limité.

Nous avons vu aussi qu'il était intéressant de distinguer plusieurs classes d'agents mobiles selon leur granularité de traitement, leurs propriétés de mobilité et de coopération. Ceci a conduit à définir un modèle de calcul d'agents mobiles coopératifs susceptible d'être structuré en couches hiérarchisées. Cette approche peut être mise en parallèle avec les couches de processus (lourds et légers) dans les systèmes d'exploitation et constitue un premier pas vers la définition de classes génériques d'agents mobiles.

Perspectives

En conclusion de notre étude, les agents mobiles apparaissent comme des entités de base qui doivent être spécialisées, structurées et composées selon des règles génériques pour en faciliter l'usage, la programmation et les performances. Un parallèle peut être ici fait avec l'étude du calcul réparti en terme de communication par messages.

Partant d'un modèle de calcul réparti événementiel fondé sur l'échange de messages, l'algorithmique répartie classique , fondée sur l'hypothèse d'un réseau de communication pratiquement stable, a cherché à concevoir des protocoles de communication de plus haut niveau pour faciliter l'expression des algorithmes répartis. À titre d'exemple, on citera la notion de jeton circulant, de calcul diffusant, d'arbre de recouvrement, d'observation, de vague, de groupe, etc

Les agents mobiles constituent à leur tour la brique élémentaire sur laquelle peut reposer un calcul réparti. Pour en exploiter toute la richesse et en faciliter la programmation, la mise en œuvre et l'exploitation, de nouvelles abstractions ou lois de composition doivent être élaborées pour constituer une base algorithmique répartie centrée sur l'usage des agents mobiles. Nos

travaux s'inscrivent dans cette démarche générale et constituent une première contribution. Cependant, beaucoup reste à faire, en particulier :

- en terme de classes génériques d'agents mobiles caractérisées par des propriétés de granularité, de mobilité, de coopération, de sécurité, d'auto-stabilisation, de tolérance aux fautes, etc
- en terme de services de base nécessaires dans tout calcul d'agents mobiles : par exemple, service de désignation et localisation, service de régulation, service de sécurité, etc
- en terme de lois de composition entre agents et/ou groupes d'agents de façon à garantir par construction que l'assemblage de différentes classes ou couches hiérarchisées d'agents reste cohérent.
- en terme de modèle de calcul générique fondé sur des agents mobiles. De façon similaire à la notion de calcul diffusant, on peut par exemple définir un calcul d'agents à mobilité aléatoire et coopérant uniquement par rencontre sur un même site. Il s'agira alors d'étudier sous quelles conditions un tel calcul peut atteindre un état stable (se terminer par exemple) ou être auto-stabilisant.

Des travaux importants restent donc à faire pour exploiter correctement les possibilités offertes par la technologie « agents mobiles à large échelle » dans le contexte de plus en plus fréquemment répandu des réseaux de communication dynamiques.

Bibliographie

[ADB+99] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. *Lecture Notes in Computer Science*, 1707 :304–307, 1999. 55

[ALP04] Jean-Paul Arcangeli, Sébastien Leriche, and Marc Pantel. Development of Flexible Peer-to-Peer Information Systems using Adaptable Mobile Agents. In *1st Int. Workshop on Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems (GLOBE'04), Zaragoza, Spain*, pages 549–553. IEEE Computing Society, 30 août-3 septembre 2004. 41

[Ant04] Terho Antila. Internetworking with SOAP. In *Seminar on Internetworking 2004 (3cr)*. Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology, May 27 2004. 17

[ARB03] Joan Ametller, Sergi Robles, and Joan Borrell. Agent migration over FIPA ACL messages. In *MATA*, pages 210–219, 2003. 27, 40

[AS92] Jean-Marc Alliot and Thomas Schiex. *Intelligence Artificielle et Informatique Theorique*. Collection Intelligence Artificielle. Cépaduès Edition, 1992. 21

[BB02] Mohammed Benattou and Jean-Michel Bruel. Active objects for coordination in distributed testing. *Lecture Notes in Computer Science*, 2425 :348–357, 2002. 27

[BBL02] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and grid technologies for wide-area distributed computing. *International Journal of Software : Practice and Experience (SPE)*, 32(15) :1437–1466, December 2002. 55

[BCL+04] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. An open component model and its support in java. In Ivica Crnkovic, Judith A. Stafford, Heinz W. Schmidt, and Kurt C. Wallnau, editors, *CBSE*, volume 3054 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2004. 130

[BCTR05] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, and Giovanni Rimassa. *JADE Programmer's Guide*. TILab S.p.A., http ://jade.tilab.com, mars 2005. 30

[BDDN01] Marc Bui, Sajal K. Das, Ajoy Kumar Datta, and Dai Tho Nguyen. Randomized mobile agent based routing in wireless networks. *International Journal of Foundations of Computer Science*, 12(3) :365–384, 2001. 106, 119

[BDGO03] C. Bertelle, A. Dutot, F. Guinand, and D. Olivier. Simulations distribuées par un algorithme fourmi. In *Rencontres Francophones du Parallélisme (RenPar)*, pages 56–63, La Colle sur Loup, France, 15-17 octobre 2003. 106

[Ber99a] Guy Bernard. Applicabilité et performances des systèmes d'agents mobiles dans les systèmes répartis. In *Première Conférence Française en Systèmes d'Exploitation (CFSE'1)*, Rennes, France, Juin 8-11 1999. 24

[Ber99b] Guy Bernard. Technologie du code mobile : état de l'art et persecptive. In *Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'99)*, Nancy, France, Avril 29-29 1999. 24

[Ber00] Guy Bernard. Apport des agents mobiles à l'exécution répartie. In *4ème Ecole d'Informatique des Systèmes Parallèles et Répartis (ISYPAR'00)*, Toulouse, France, fevrier 2000. 33

[BFM98] Petra Berenbrink, Tom Friedetzsky, and Ernst W. Mayr. Parallel continuous randomized load balancing. In *Proceedings of the tenth annual ACM symposium on Parallel algorithms and architectures*, pages 192–201, New York, NY, USA, 1998. ACM Press. 122

[BHK+04] S. Bouchenak, D. Hagimont, S. Krakowiak, N. De Palma, and F. Boyer. Experiences implementing efficient Java thread serialization, mobility and persistence. *Software - Practice and Experience*, 34(4) :355–393, April 2004. 20

[BHR97] Joachim Baumann, Fritz Hohl, and Kurt Rothermel. Mole - concepts of a mobile agent system. Technical Report 1997/15, Universität Stuttgart - Fakultät Informatik, August 29 1997. 34

[BHV01] Walter Binder, Jarle G. Hulaas, and Alex Villaz. Portable resource control in the J-SEAL2 mobile agent system. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 222–223, Montreal, Canada, May 2001. ACM Press. 34

[BK04] Badr Benmammar and Francine Krief. La technologie agent et les réseaux sans-fil. HAL - CCSd - CNRS, November 28 2004. 20

[BKR98] Jonathan Bredin, David Kotz, and Daniela Rus. Utility Driven Mobile-Agent Scheduling. Technical Report PCS-TR98-311, Dartmouth College, Computer Science, Hanover, NH, may 1998. 46

[BLD02] Lorenzo Bettini, Nicola Michele Loreti, and Rocco De. Formalizing properties of mobile agent systems. In *Coordination Models and Languages*, pages 72–87, January 29 2002. 23

[BN84] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1) :39–59, February 1984. 16

[Bor94] Nathaniel S. Borenstein. E-mail with a mind of its own : The Safe-Tcl language for enabled mail. *IFIP Transactions C. Comm SydLA*, 25 :389–402, 1994. 34

[BPR99] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE — A FIPA-compliant agent framework. In *Proceedings of the 4th International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-99)*, pages 97–108, London, UK, 1999. The Practical Application Company Ltd. 27, 39, 41

[BZS93] B. N. Bershad, M. J. Zekauskas, and W. A. Sawdon. The midway distributed shared memory system. In *Proceedings of Compcon '93*, pages 528–537, San Francisco, February 1993. IEEE. 14

[Car94] Luca Cardelli. Obliq : A language with distributed scope. Research Report 122, Digital Equipment Corporation Systems Research Center, 1994. 34

[CCD+02] Pascal Chour, Frédéric Cuppens, Yves Deswarte, Ludovic Mé, Refik Molva, and Yves Roudier. *Sécurité des réseaux et systèmes répartis*. Hermes Science, 2002. 27

[CD98] G. Caro and M. Dorigo. Antnet : Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9 :317–365, 1998. 52, 72

[CHK94] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile Agents : Are They a Good Idea ? Technical Report RC 19887, IBM Research Division, T.J. Watson Research Center, 1994. 24

[CHK+00] S. Campadello, H. Helin, O. Koskimies, P. Misikangas, M. Mäkelä, and K. Raatikainen. Using mobile and intelligent agents to support nomadic users. In *Proceedings of the 6th International Conference on Intelligence in Networks (ICIN2000)*, pages 199–204, Bordeaux, France, January 2000. 57

[Ciz98] Gisèle Cizault. *IPv6 : Théorie et pratique*. O'REILLY, 1998. 9, 63

[CLMZ01] Giacomo Cabri, Letizia Leonardi, Marco Mamei, and Franco Zambonelli. Engineering infrastructures for mobile organizations. *Lecture Notes in Computer Science*, 2203 :39–56, 2001. 45

[CM99] S. Corson and J. Macker. Mobile ad hoc networking (MANET) : Routing protocol performance issues and evaluation considerations. Request For Comments (RFC) 2501, January 1999. statut : « Informational », http ://www.ietf.org/rfc/rfc2501.txt. 12

[Col97] Jean-Louis Colaço. *Analyses Statiques de Langages d'Acteurs par inférence de types*. Thése de doctorat, ENSEEIHT, Toulouse, octobre 1997. 27

[Com96] Douglas Comer. *TCP/IP Architecture, protocoles, applications*. InterEdition, Paris, 1996. 7

[CPQ05] Michel Charpentier, Gérard Padiou, and Philippe Quéinnec. Cooperative mobile agents to gather global information. In *4th IEEE Int'l Symposium on Network Computing and Applications*, pages 271–274, July 2005.

[CPV97] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Designing Distributed Applications with Mobile Code Paradigms. In R. Taylor, editor, *Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, pages 22–32. ACM Press, 1997. 17

[Cub01] Christophe Cubat Dit Cros. Supervision d'agents mobiles à travers une application intra-internet. Mémoire de DEA, ENSEEIHT, jun 2001. 55

[Cub04] Christophe Cubat Dit Cros. Agents mobiles coopératifs pour les environnements dynamiques. In *Les Nouvelles Technologies de la Répartition (NOTERE'2004)*, Saidia, Maroc, juin 2004.

[CWKS97] Brian P. Crow, Indra Widjaja, Jeong Geun Kim, and Prescott T. Sakai. IEEE 802.11 : Wireless local area networks. *IEEE Communications Magazine*, 35(9) :116–126, 09 1997. 11

[D+87] Alan Demers et al. Epidemic algorithms for replicated database maintenance. In *6th Symposium on Principles of Distributed Computing*, pages 1–12, August 1987. 39, 107

[DH98] F. Dawson and T. Howes. RFC 2426 : vCard MIME directory profile. ftp ://ftp.internic.net/rfc/rfc2426.txt, September 1998. Status : PROPOSED STANDARD. 135

[Dij74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11)(11) :643–644, 1974. 122

[Dom96] Peter Domel. Mobile Telescript agents and the Web. In *Digest of Papers. COMPCON '96. Technologies for the Information Superhighway. Forty-First IEEE Computer Society International Conference*, pages 52–57. IEEE Computer Society Press, 2 1996. 34, 41

[DRL98] Jocelyn Desbiens, Francis Renaud, and Martin Lavoie. Communication and tracking infrastructure of a mobile agent system. In *Thirty-First Annual Hawaii International Conference on System Sciences*, volume 7, pages 54–63, January 1998. 63

[DS84] P. G. Doyle and J. L Snell. *Random Walks and Electric Networks*. Number 22 in Carus Mathematical Monographs. Mathematical Association of America, 1984. 48

[DSW02] Shlomi Dolev, Elad Schiller, and Jennifer Welsh. Random walk for self-stabilizing group communication in ad hoc networks. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing (PODC'02)*, pages 259–259, New York, NY, USA, 2002. ACM Press. 119

[Dun01] Cameron Ross Dunne. Using mobile agents for network resource discovery in peer-to-peer networks. *Special Interest Group on Electronic Commerce (SIGecom) Exch.*, 2(3) :1–9, 2001. 14, 54, 77

[Fer95] Jacques Ferber. *Les Systèmes Multi-Agents*. Informatique Intelligence Articielle. InterEdition, 1995. 19, 21, 44

[FIP02] Foundation for Intelligent Physical Agents (www.fipa.org). *FIPA Abstract Architecture Specification*, December 2002. statut : « Standard, version L ». 35

[FPV98] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5) :342–361, May 1998. 19, 20, 33, 86

[Gau04] Picard Gauthier. *Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalités émergentes*. Thèse de doctorat, Université Paul Sabatier de Toulouse III, décembre 2004. 44

[GCK+02] Robert S. Gray, George Cybenko, David Kotz, Ronald A. Peterson, and Daniela Rus. D'agents : Applications and performance of a mobile-agent system. *Softw., Pract. Exper.*, 32(6) :543–573, 2002. 24, 41

[GCKR98] Robert S. Gray, George Cybenko, David Kotz, and Daniela Rus. D'Agents : Security in a multiple-language, mobile-agent system. In Giovanni Vigna, editor, *Mobile Agent Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 154–187. Springer-Verlag : Heidelberg, Germany, 1998. 30

[GCKR00] R. S. Gray, G. Cybenko, D. Kotz, and D. Rus. Mobile agents : Motivations and state of the art. Technical Report TR2000-365, Department of Computer Science, Dartmouth College, Hanover, New Hampshire, USA, April 2000. 144

[GCKR01] R. S. Gray, G. Cybenko, D. Kotz, and D. Rus. *Mobile agents : Motivations and State of the Art*. AAAI/MIT-Press, 2001. publié aussi dans [GCKR00]. 25

[Gel85] David Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1) :80–112, January 1985. 14

[GGM99] Jean-Marc Geib, Christophe Gransart, and Philippe Merle. *CORBA : des concepts à la pratique*. Editions Dunod, Paris, France, Octobre 1999. 17, 26, 77, 130

[Gie78] Michel Gien. File Transfer Protocol (ftp). *Computer Networks : The International Journal of Distributed Informatique*, 2(4–5) :312–319, September/October 1978. 16

[GMS04] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walk in peer-to-peer networks. In *IEEE INFOCOM, The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, Hong Kong, China, March 7-11 2004. 119

[GOP02] R.H Glitho, E. Olougouna, and S. Pierre. Mobile agents and their use for information retrieval : a brief overview and an elaborate case study. *IEEE Network Magazine*, 16 :34–41, Jan/Feb 2002. 133

[Gra59] Pierre-Paul Grassé. La reconstruction du nid et les coordinations inter-individuelles chez *Belicositermes natalensis et Cubitermes sp*. la théorie de la stigmergie : Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6 :41–80, 1959. 106

[Gra04] Robert S. Gray. Mobile agents : Overcoming early hype and a bad name. In *5th IEEE International Conference on Mobile Data Management (MDM 2004)*, pages 302–303, Berkeley, CA, USA, 19-22 January 2004. IEEE Computer Society. 32

[HGB04] Luc Hogie, Frédéric Guinand, and Pascal Bouvry. A heuristic for efficient broadcasting in the metropolitan ad hoc networks. In Mircea Gh. Negoita, Robert J. Howlett, and Lakhmi C. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems (KES), 8th International Conference*, volume 3213 of *Lecture Notes in Computer Science*, pages 727–733, Wellington, New Zealand, September 20-25 2004. Springer. 13

[HMW98] Martin O. Hofmann, Amy McGovern, and Kenneth R. Whitebread. Mobile agents on the digital battlefield. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 219–225, New York, May 9–13, 1998. ACM Press. 12

[IH99] Leila Ismael and Daniel Hagimont. A performance evaluation of mobile agent paradigm. In *Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA' 99)*, pages 306–313, Novembre 1999. 25

[ITK99] And Yasuyoshi Inagaki, Katsuhiko Toyama, and Nobuo Kawaguchi. Magnet : Adhoc network system based on mobile agents, September 30 1999. 26

[JGC04] Christine Julien and Roman G-C. Active coordination in ad hoc networks. In *Proceedings of the 6th International Conference on Coordination Models and Languages*, pages 199–215, Pisa (Italy), February 2004. 55

[JHK99] Sam Joseph, Masanori Hattori, and Naoki Kaze. Learning improves mobile agent efficiency. In *1st Asia-Pacific Conference on Intelligent Agent Technology*, pages 416–425. World Scientific, 1999. 47

[JLvR+02] Dag Johansen, Kaare J. Lauvset, Robbert van Renesse, Fred B. Schneider, Nils P. Sudmann, and Kjetil Jacobsen. A TACOMA retrospective. *Software & Practice and Experience*, 32(6) :605–619, May 2002. 38, 41

[Joh98] Dag Johansen. Mobile agent applicability. *Lecture Notes in Computer Science*, 1477 :80–98, 1998. 24

[Joh04] Dag Johansen. Mobile agents : Right concept, wrong approach. In *Mobile Data Management*, pages 300–301, 2004. 26

[Jon02] M. Tim Jones. Java mobile agents and the Aglets SDK. *Dr. Dobb's Journal of Software Tools*, 27(1) :42, 44, 46–48, January 2002. 41

[JvRS95] Dag Johansen, Robbert van Renesse, and Fred B. Schneider. Operating System Support for Mobile Agents. In *Proceedings of the Fifth Workshop Hot Topics in Operating Systems (HotOS)*, pages 42–45, May 1995. 38

[JXHX02] Lu Jun, Lu Xianlang, Han Hong, and Zhou Xu. Application of mobile agent in wide area network. In *IEEE*, 2002. 24

[KB01] Djcemal Kebbal and Guy Bernard. Component search service and deployment of distributed applications. In *3rd International Symposium on Distributed Objects and Applications (DOA'01)*, Sepember 2001. 93

[KG99] D. Kotz and R. S. Gray. Mobile agents and the future of the internet. *Operating Systems Review*, 33(3) :7–13, July 1999. 24

[KGR02] D. Kotz, R. S. Gray, and D. Rus. Future directions for mobile-agent research. Technical Report TR2002-415, Department of Computer Science, Dartmouth College, Hanover, New Hampshire, USA, January 2002. 32

[KKPS98] Elizabeth A. Kendall, P. V. Murali Krishna, Chirag V. Pathak, and C. B. Suresh. Patterns of intelligent and mobile agents. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 92–99, New York, May 9–13, 1998. ACM Press. 22

[Kna96] Frederick C. Knabe. An overview of mobile agent programming. In *Proceedings of the 5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages*, Stockholm, Sweden, June 1996. 33

[KNY95] Naoki Kobayashi, Motoki Nakade, and Akinori Yonezawa. Static analysis of communication for asynchronous concurrent programming languages. In *Second International Static Analysis Symposium (SAS'95)*, volume 983, pages 225–242. Springer-Verlag, 1995. 27

[KTI99] Nobuo Kawaguchi, Katsuhiko Toyama, and Yasuyoshi Inagaki. MAGNET : Ad-hoc network system based on mobile agents. In Ahmed Karmouch and Roger Impley, editors, *First International Workshop on Mobile Agents for Telecommunication Applications (MATA'99)*, pages 127–142. World Scientific Publishing Ltd., 10 1999. 12

[LA03] Anis Laouiti and Cédric Adjih. Mesures des performances du protocole OLSR. In *IEEE SETIT2003*, Tunisia, March 2003. 12

[Lam02] L Lamport. *Specifying Systems : The TLA+ language and tools for Hardware and Software Engineers*. Addison Wesley Professional, 2002. 122

[Lao02] Anis Laouiti. *Unicast et Multicast dans les réseaux ad hoc sans fil*. PhD thesis, Université de Versailles Saint-Quentin-En-Yvelines, Juillet 2002. 10

[LJJM01] Kåre J. Lauvset, Kjetil Jacobsen, Dag Johansen, and Keith Marzullo. Separating mobility from mobile agents. In *Proceedings of HotOS-VIII : 8th Workshop on Hot Topics in Operating Systems*, page 173. IEEE Computer Society, May 20-23, 2001. 22

[LK02] Mulugeta Libsie and Harald Kosch. Content adaptation of multimedia delivery and indexing using MPEG-7. In *Proceedings of the tenth ACM international conference on Multimedia (MM-02)*, pages 644–646, New York, December 1–6 2002. ACM Press. 26

[Lou01] S. Loureiro. *Mobile Code Protection*. Thèse de doctorat, ENST Paris / Institut Eurecom, 2001. 28, 29

[LSC+85] B. Lyon, G. Sager, J. M. Chang, D. Goldberg, S. Kleiman, T. Lyon, R. Sandberg, D. Walsh, and P. Weiss. Overview of the Sun network file system. Technical Report CMU-CS-84-137, Sun Microsystems, Inc., January 1985. 16

[Lyu95] Michael R. Lyu. *Software Fault Tolerance*. John Wiley & sons Ltd, 1995. 122

[MAL] Mobile Agent Community. *The Mobile Agent List*. http ://mole.informatik.uni-stuttgart.de/mal/mal.html. 26, 33

[MBB+98] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF : The OMG Mobile Agent System Interoperability Facility. In K. Rothermel and F. Hohl, editors, *Proceedings of the 2nd International Workshop on Mobile Agents*, volume 1477 of *Lecture Notes in Computer Science*, pages 50–67. Springer-Verlag : Heidelberg, Germany, 1998. 21, 35

[MDP+00] D.S. Milojǐcić, F. Douglis, Y. Paindaveine, R. Weeler, and S. Zhou. Process migration. *ACM Compting Surveys*, 32(3) :241–299, Septembre 2000. 19

[MDW99] D. Milojicic, F. Douglis, and R. Wheeler. *Mobility : processes, computers and agents*. Addison-Wesley, 1999. 18, 22, 25, 86

[Mey97] Bertrand Meyer. *Object-Oriented Software Construction*. The Object-Oriented Series. Prentice-Hall, Englewood Cliffs (NJ), USA, 2nd edition, 1997. 16

[Mil99] Dejan Milojicic. Mobile agent applications. *IEEE Concurrency*, 7(3) :80–90, July/September 1999. 32

[Mis99] Jeff Mischkinsky. Corba 3.0 new components chapters. TC Document - CCM FTF Draft ptc/99-10-04, OMG (Object Management Group), Octobre 1999. 128

[MKM99] Nelson Minar, Kwindla Hultman Kramer, and Pattie Maes. Cooperating mobile agents for mapping networks. In *Proceedings of the First Hungarian National Conference on Agent Based Computation*, 1999. 50, 55, 98

[MP00] R. Marvie and M. Pellegrini. Etat de l'art des modèles de composants. Technical Report RNRT 98 CESURE-2, Projet CESURE, Mai 2000. 129

[MPR00] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. Coordination and Mobility. In A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors, *Coordination of Internet Agents : Models, Technologies, and Applications*, pages 254–273. Springer, 2000. Invited contribution. 49

[MPR01] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. Lime : A middleware for physical and logical mobility. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'2001)*, pages 524–533, Mesa, Arizona, USA, April 2001. 36

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge International Series in Parallel Computation. Cambridge University Press, 1995. 48

[MR98] P. J. McCann and G. C. Roman. Compositional programming abstractions for mobile computing. *IEEE Transactions on Software Engineering*, 24(2) :97–110, 1998. 23

[MS99] Manuel de Jesus Mendes and Flávio M. de Assis Silva. Mobile agents in autonomous decentralized systems. In *ISADS*, pages 258–260, 1999. 32, 54

[MSS+01] Paulo Marques, Paulo Simões, Luís M. Silva, Fernando Boavida, and João G. Silva. Providing applications with mobile agent technology. In *OpenArch'01 - Fourth IEEE International Conference on Open Architectures and Network Programming*, Anchorage, Alaska, April 2001. 54

[MX03] Shivakant Mishra and Peng Xie. Interagent communication and synchronization support in the DaAgent mobile agent-based computing system. *IEEE Trans. Parallel and Distrib. Systems*, 14(3) :290–306, 2003. 48

[Nel81] Bruce Jay Nelson. *Remote procedure call*. PhD thesis, Carnegie-Mellon University, 1981. 16

[NL98] George C. Necula and Peter Lee. Safe, untrusted agents using proof-carrying code. In Giovanni Vigna, editor, *Mobile Agent Security*, Lecture Notes in Computer Science No. 1419, pages 61–91. Springer-Verlag : Heidelberg, Germany, 1998. 29

[Obj03] Object Management Group, Inc. *Unified Modeling Language (UML) 2.0 Superstructure Specification*, aug 2003. Final Adopted Specification. 23

[ONI+97] Akihiko Ohsuga, Yasuo Nagai, Yutaka Irie, Masanori Hattori, and Shinichi Honiden. PLANGENT : an approach to making mobile agents intelligent. *IEEE Internet Computing*, 1(4) :50–57, July/August 1997. 37, 41

[PC02] Gian Pietro Picco and Gianpaolo Cugola. Peer-to-peer for collaborative applications. In *International Workshop on Mobile Teamwork Support, co-located with the 22nd International Conference on Distributed Computing Systems*, pages 359–364, Vienna (Austria), July 2002. IEEE press. 14

[Pic98] Gian Pietro Picco. *Understanding, Evaluating, Formalizing, and Exploiting Code Mobility*. PhD thesis, Politecnico di Torino, Italy, February 1998. 17, 26

[Pic01] Gian Pietro Picco. Mobile Agents : An Introduction. *Journal of Microprocessors and Microsystems*, 25 :65–74, 2001. Invited contribution to a special issue on mobile agents. 15

[PMR99] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. Lime : Linda meets mobility. In *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, pages 368–377, Los Angeles, California, USA, May 1999. http ://www.cs.rochester.edu/u/murphy/papers/icse99.pdf. 36, 41

[PQMC05] Gérard Padiou, Philippe Quéinnec, Philippe Mauran, and Christophe Cubat Dit Cros. Composants mobiles fondés sur des agents mobiles coopérants. In *Journées Composants*, pages 80–86, Le Croisic, France, 5-8 avril 2005. ASF : Association ACM-SIGOPS de France.

[PRT+03] Agostino Poggi, Giovanni Rimassa, Paola Turci, James Odell, Haralambos Mouratidis, and Gordon A. Manson. Modeling deployment and mobility issues in multiagent systems using AUML. In *AOSE*, pages 69–84, 2003. 23

[PS93] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. In *Proceedings 8th IEEE Logics in Computer Science*, pages 376–385, Montreal, Canada, 1993. 27

[PS97] Holger Peine and Torsten Stolpmann. The architecture of the Ara platform for mobile agents. In Radu Popescu-Zeletin and Kurt Rothermel, editors, *First International Workshop on Mobile Agents MA'97*, Berlin, Germany, 1997. 41

[Ran75] B. Randell. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, SE-1(2) :220–232, june 1975. 95

[RB05] Fabien Romeo and Franck Barbier. Administration de composants logiciels : application aux systèmes sans fil. *revue Génie Logiciel, GL & IS*, 73 :39–43, 2005. 129

[RCBL04] Amar Ramdane-Cherif, Samir Benarif, and Nicole Lévy. Agent technologies for pervasive computing and communications. In Christine W. Chan, Witold Kinsner, Yingxu Wang, and D. Michael Miller, editors, *Proceedings of the 3rd IEEE International Conference on Cognitive Informatics (ICCI 2004)*, pages 230–239, Victoria, Canada, 16-17 August 2004. IEEE Computer Society. 80

[RDP00] Marcelo Goncalves Rubinstein, Otto Carlos Muniz Bandeira Duarte, and Guy Pujolle. Improving management performance by using multiple mobile agents. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 165–166, Barcelona, Catalonia, Spain, June 2000. ACM Press. 55

[RG91] Deborah Russell and G. T. Gangemi Sr. *Computer Security Basics*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1991. A clear overview on many different security issues. 27

[Rif98] Jean-Marie Rifflet. *La Programmation sous Unix*. Ediscience International, Paris, 1998. 16

[RM00] Michel Riveill and Philippe Merle. *La programmation par composants*. Techniques de l'Ingénieur - Informatique, 2000. 129

[Rol99] Christian Rolland. *LaTeX par la pratique*. O'REILLY, Paris, 1999. ii

[Rot04] Volker Roth. Obstacles to the adoption of mobile agents. In *5th IEEE International Conference on Mobile Data Management (MDM 2004)*, pages 296–297. IEEE Computer Society, 19-22 January 2004. 28

[Rou02] Siegfried Rouvrais. *Utilisation d'Agents Mobiles pour la Construction de Services Distribués*. Thèse, Université de Rennes 1, 2002. 14, 25, 35

[San04] Jean-Paul Sansonnet. Introduction aux systèmes multi-agents. Cours SMA en ligne du Master recherche de Paris XI, 2004. 21

[Sat00] Ichiro Satoh. A formalism for hierarchical mobile agents. In *International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 2000)*, pages 165–172, Limerick, Ireland, 10-11 June 2000. IEEE Computer Society. 52

[Sat02] Ichiro Satoh. Physical mobility and logical mobility in ubiquitous computing environments. *Lecture Notes in Computer Science*, 2535 :186–202, 2002. 26

[Sat04a] Ichiro Satoh. Linking physical worlds to logical worlds with mobile agents. In *5th IEEE International Conference on Mobile Data Management (MDM 2004)*, page 332, Berkeley, CA, USA, 19-22 January 2004. IEEE Computer Society. 55

[Sat04b] Ichiro Satoh. Selection of mobile agents. In *24th International Conference on Distributed Computing Systems (ICDCS 2004)*, pages 484–493, Hachioji, Tokyo, Japan, 24-26 March 2004. IEEE Computer Society. 52

[SBB+00] Niranjan Suri, Jeffrey M. Bradshaw, Maggie R. Breedy, Paul T. Groth, Gregory A. Hill, Renia Jeffers, Timothy S. Mitrovich, Brian R. Pouliot, and David S. Smith. Nomads : toward a strong and safe mobile agent system. In *AGENTS '00 : Proceedings of the fourth international conference on Autonomous agents*, pages 163–164. ACM Press, 2000. 34

[Shi00] Clay Shirky. What is p2p ... and what isn't. O'Reilly Network, November 2000. 14

[SHK95] Behrooz A. Shirazi, Ali R. Hurson, and Krishna M. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press - John Wiley & Sons, Inc., March 1995. 120

[SM98] Akhil Sahai and Christine Morin. Mobile agents for enabling mobile user aware applications. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 205–211, New York, May 9–13, 1998. ACM Press. 24, 47

[Spi76] F. Spitzer. *Principles of Random Walks*. Graduate Texts in Mathematics. Springer-Verlag ; 2d edition, 1976. 48

[SS04] Alan Schmitt and Jean-Bernard Stefani. The kell calculus : A family of higher-order distributed process calculi. In *Global Computing*, pages 146–178, 2004. 23

[SWP99] Peter Sewell, Paweł T. Wojciechowski, and Benjamin C. Pierce. Location-independent communication for mobile agents : A two-level architecture. *Lecture Notes in Computer Science*, 1686 :1+, 1999. 23

[Tan90] Andrew Tanenbaum. *RÉSEAUX : Architecture, protocoles, applications*. InterÉditions, Paris, 1990. 7

[TBV04] Andrew S. Tanenbaum, Frances M. T. Brazier, and Guido J. Van 't Noordende. Security in a mobile agent system. citeseer in line, July 06 2004. http ://citeseer.ist.psu.edu/705581.html. 30

[Tho97] Tommy Thorn. Programming languages for mobile code. *ACM Computing Surveys*, 29(3) :213–239, September 1997. 17, 33

[TOH01] Yasuyuki Tahara, Akihiko Ohsuga, and Shinichi Honiden. Mobile agent security with the IPEditor development tool and the mobile UNITY language. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 656–662, Montreal, Canada, May 2001. ACM Press. 30

[TS04] Hui Tian and Hong Shen. Mobile agents based topology discovery algorithms and modelling. In *ISPAN*, pages 502–507, 2004. 52

[TVP00] Orazio Tomarchio, Lorenzo Vita, and Antonio Puliafito. Nomadic users' support in the map agent platform. *Lecture Notes In Computer Science*, 1931 :233–242, 2000. Proceedings of the Second International Workshop on Mobile Agents for Telecommunication Applications. 57

[TWB03] Arnaud Troël, Frédéric Weis, and Michel Banâtre. Découverte automatique entre terminaux mobiles communicants. Technical Report PI-1570, IRISA, octobre 2003. 64

[UK99] Andreas Ulrich and Hartmut König. Architectures for testing distributed systems. In *Testing of Communicating Systems : Method and Applications, IFIP TC6 12th International Workshop on Testing Communicating Systems (IWTCS)*, IFIP Conference Proceedings, pages 93–108, Budapest, Hungary, September 1-3, 1999. Kluwer. 27

[VB00] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks, May 08 2000. 12

[Vig04] Giovanni Vigna. Mobile agents : Ten reasons for failure. In *5th IEEE International Conference on Mobile Data Management (MDM 2004)*, pages 298–299. IEEE Computer Society, 19-22 January 2004. 26, 28

[VM04] Luminita Vasiu and Qusay H. Mahmoud. Mobile agents in wireless devices. *IEEE Computer*, 37(2) :104–105, 2004. 57

[Whi94] J. E. White. Telescript technology : The foundation for the electronic marketplace. White paper, General Magic, Inc., 2465 Latham Street, Mountain View, CA 94040, 1994. 19

[WMC04] James L. Weaver, Kevin Mukhar, and Jim Crume. *J2EE 1.4*. Editions Eyrolles, Sepembre 2004. 128

[Zac03] John Zachary. Protecting mobile code in the wild. *IEEE Internet Computing*, 7(2) :78–82, 2003. 28, 29

[Zas04] Arkady B. Zaslavsky. Mobile agents : Can they assist with context awareness ? In *5th IEEE International Conference on Mobile Data Management (MDM 2004)*, pages 304–305. IEEE Computer Society, 19-22 January 2004. 55

Table des figures

Liste des tableaux

Index

Glossaire

ACL : Agents Communication Language
AUML : Agent UML

FIPA : Foundation for Intelligent Physical Agents

internet : Interconnection Network

JADE : Java Agents DEvelopment Framework

LIME : Linda In Mobile Environnement

MANET : Mobile Ad'hoc NETwork
MASIF : Mobile Agent System Interoperability Facility

PCC : Proof Carrying Code
PLANGENT : Plan Agent

SMA : Système Multi-Agents

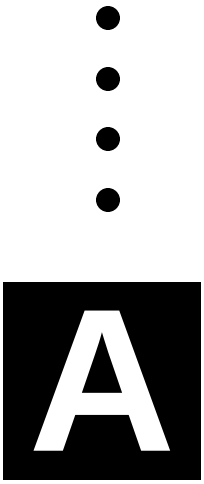
TACOMA : Troms And COrnell Moving Agents
TCP/IP : Transmission Control Protocol et Internet Protocol

UML : Unified Modeling Language

Quatrième partie

Annexes

Simulateur



Dans cette annexe nous présentons le simulateur que nous avons réalisé en Java. Son but est de mettre en place les environnements distribués dynamiques et le modèle d'agents mobiles coopérants que nous avons présentés dans ce documents. Grâce à la simulation du contexte, nous pouvons tester notre modèle en faisant varier un certain nombres de critères pour observer les phénomènes exposés dans le chapitre décrivant les expérimentations.

Cette annexe comprend deux parties. La première porte sur les objectifs du simulateur, à savoir l'environnement simulé, ses paramètres et sa généricité. La seconde porte sur la description effective du simulateur en commençant par le diagramme UML correspondant. Nous regardons, ensuite, l'adéquation avec notre modèle d'agents mobiles coopérants et nous terminons par l'exposé des principes de fonctionnement du simulateur.

A.1 Objectif du simulateur

A.1.1 L'environnement simulé

L'environnement simulé doit comporter la double mobilité, physique et logicielle, que nous avons présentée dans le chapitre 3. Pour la mobilité physique, nous utilisons une classe « *Noeud* » qui représente un site pouvant se déplacer. Chaque *Noeud* possède un ensemble de voisins avec lesquels il peut communiquer directement. Cet ensemble peut être vide et nous parlons alors de *Noeud* isolé. La mobilité physique est obtenue en modifiant cet ensemble de voisins.

Notons que cette méthode correspond aux environnements hybrides dynamiques que nous envisageons car, d'un point de vue des sites, la mobilité physique n'est pas considérée en terme de localisation mais bien en fonction du voisinage accessible. Autrement dit, un site ne considère pas qu'il est en mouvement mais perçoit que son environnement local change dynamiquement. Un site a donc bien la charge uniquement de son contexte local.

Pour ce qui est de la mobilité logicielle, nous définissons une classe « *Agent* » qui représente un agent mobile. Chaque agent est rattaché à un *Noeud* lors de sa création. Pour obtenir la migration d'un agent, on modifie directement le noeud de rattachement en choisissant un des *Noeud* appartenant à l'ensemble de voisins. Ainsi, la migration est bien effectuée entre deux *Noeud* de l'environnement qui possèdent un lien de communication directe. Un agent ne peut pas se déplacer entre deux sites qui ne sont pas voisins.

D'un autre côté, l'environnement simulé doit garantir que toute communication entre agents se fait localement et que toute communication distante est interdite, à l'exception du service de migration. Pour cela, les agents peuvent communiquer uniquement avec les agents ayant le même `Noeud` de rattachement, garantissant ainsi la communication purement locale. Pour la migration, nous avons vu qu'il s'agit simplement d'un changement du `Noeud` de rattachement. L'agent obtient un nouvel identifiant de `Noeud` grâce au service de migration décrit dans le modèle.

A.1.2 Paramètres et généricité

Pour observer l'adéquation de notre modèle d'agent à différents types d'environnements, notre simulateur dispose d'un ensemble de paramètres que nous présentons ici de manière non-exhaustive. Ces paramètres sont d'ordre *général*, applicables aux **nœuds** ou propres aux *agents*.

duree Il s'agit d'un paramètre général qui définit la durée globale de la simulation. En faisant varier cette durée, nous pouvons chercher à savoir si le système simulé finit par atteindre un état stable.

periodeMigrateur C'est une valeur temporelle qui permet de définir la période de modification des ensembles de voisins des nœuds. Ce paramètre nous permet de simuler des environnements plus ou moins dynamiques au niveau physique en jouant sur fréquence de la modification du contexte local de chaque nœud. Ainsi nous pouvons regarder à quel degré de dynamisme physique notre modèle s'adapte le mieux.

maxVoisins Ce paramètre représente la nombre maximal de voisins d'un nœud. En modifiant l'arité maximale des sites, nous pouvons simuler un environnement plus ou moins connecté et ainsi observer si notre modèle s'adapte à des environnements faiblement couplés ou au contraire nécessite des contextes denses.

sejour Il s'agit de la durée maximale du temps de visite d'un agent. Chaque agent entame une visite dont la durée est inférieure ou égale à *sejour*, il s'agit d'un choix aléatoire. Ce paramètre nous permet de configurer la fréquence de déplacement appartenant aux critères de classification des agents (cf section 3.3 p. 71) et ainsi d'obtenir une première différenciation entre les agents lourds et légers.

dureeMigration Cette valeur permet de définir le temps de transfert standard nécessaire à une migration. La migration se faisant entre voisins, nous pouvons considérer que les performances de communication sont équitables dans l'ensemble de l'environnement. Ainsi ce paramètre sert à simuler différents volumes de données à transférer durant les migrations et de configurer un deuxième critère de classification entre les agents lourds et légers.

coopération Il s'agit d'un booléen qui permet de mettre en place le caractère coopérant des agents mobiles afin de constater l'apport de la coopération dans notre modèle.

Le dernier élément que nous pouvons modifier sont les échelles matérielle et logicielle de l'environnement simulé, à savoir le nombre de sites et d'agents. Pour cela, nous utilisons deux fichiers de noms `listeDesSites` et `listeDesUsagers` correspondant respectivement aux identifiants des noeuds et aux identifiants des agents. En remplissant plus ou moins ces fichiers, nous pouvons simuler des environnements à différente échelle.

Pour ce qui est de la généricité, nous avons choisi de définir la classe `Agent` comme la classe de base pour tout agent mobile dans le système. Cette classe est alors étendue en fonction des comportements que l'on souhaite simuler. De plus, pour obtenir le modèle coopérant, nous fournissons une interface de classe « `Coopérant` » comportant toutes les primitives définies dans le protocole de coopération. Cette interface est implantée par la classe `Agent`.

A.1.3 L'observation

Pour l'observation des simulations, nous avons choisi d'utiliser un élément extérieur au modèle, *i.e* qui n'est pas réalisé par les agents, afin de limiter les effets de l'observation sur le comportement même des agents. Cette observation se focalise sur le nombre de migrations nécessaire à un agent pour trouver un partenaire. Nous avons donc mis en place un service classique d'abonnement/publication dont le but est de calculer le nombre total de rencontres effectuées durant la simulation et la moyenne de migrations par rencontre. Nous nonmons ce service Superviseur.

Pour réaliser cette observation, chaque agent est abonné au Superviseur lors de sa création et va compter le nombre de migrations qui lui est nécessaire pour engager une coopération. Pour cela, l'agent choisit un partenaire, puis compte le nombre de migrations qu'il effectue pour l'atteindre. Lorsqu'il engage la coopération, il envoie un événement avec le nombre total de migrations au superviseur. Celui incrémente alors le nombre global de migrations et met à jour la moyenne globale de migration par rencontre.

Nous pouvons noter qu'au niveau des agents, l'observation se réalise par un maximum de deux instructions par visite d'un site. Soit la coopération n'est pas réalisable, l'agent incrémente simplement son compteur de migration, soit il publie ce compteur au Superviseur et le remet à zéro pour la prochaine coopération. Cette méthode nous permet de limiter les effets de perturbation, dus à l'observation, sur le comportement général des agents.

A.2 Architecture et comportement du simulateur

A.2.1 Diagramme UML

Dans le diagramme UML de la figure A.1, nous retrouvons tous les éléments que nous avons présentés précédemment, à savoir les classes `Noeud` et `Agent`, l'interface `Coopérant` permettant de mettre en place le modèle et l'environnement dynamique. On retrouve aussi les éléments nécessaires à l'observation, c'est à dire l'interface `Observable` implantée par *Agent* et la classe `Superviseur`.

Nous pouvons aussi voir deux classes supplémentaires. Celle nommée `Migrateur` a la charge de simuler le dynamisme physique. Pour cela, elle modifie de manière périodique, en fonction du paramètre `periodeMigrateur`, le voisinage d'un des noeuds de l'environnement en utilisant une méthode de classe de `Noeud`.

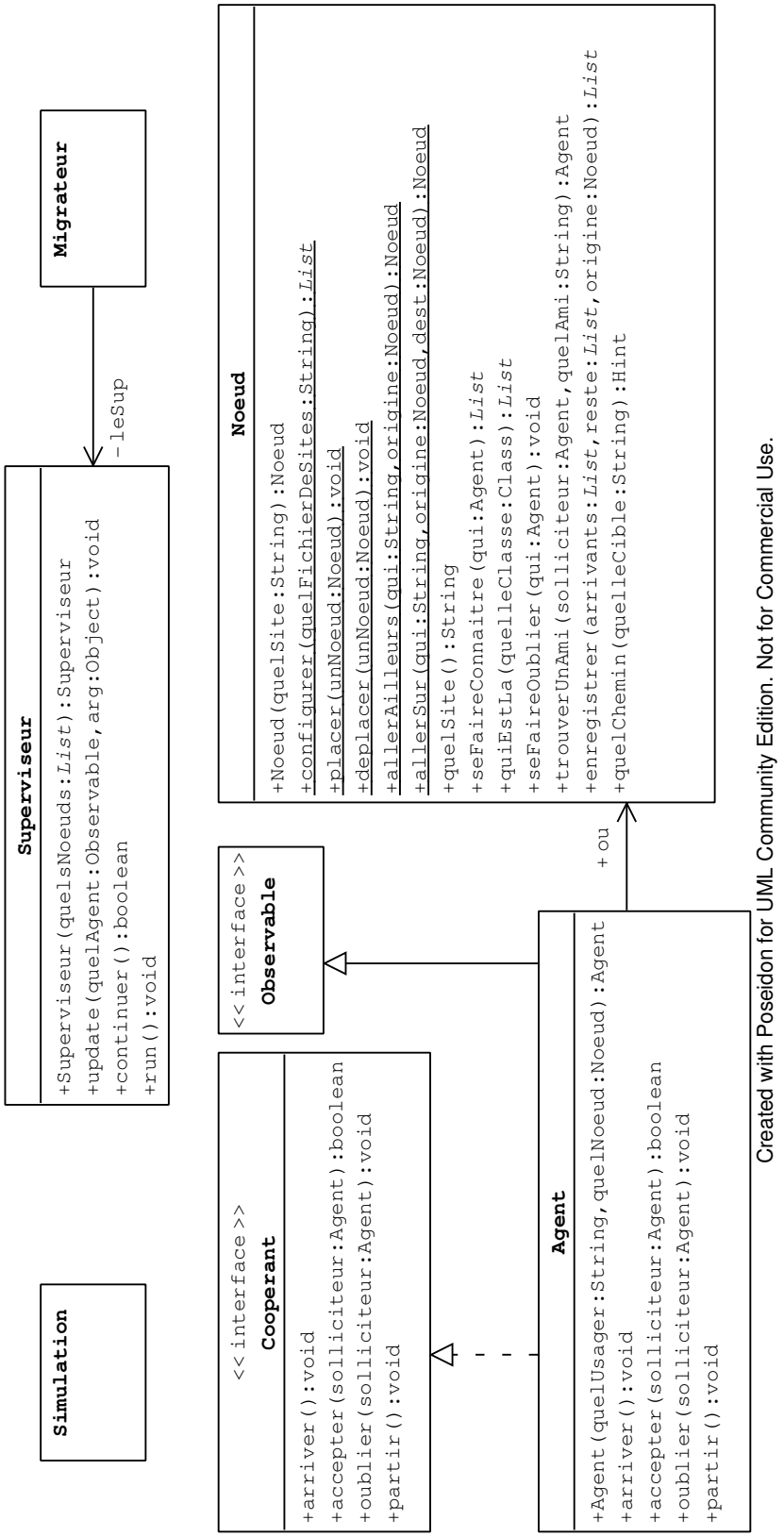


FIG. A.1 – Diagramme UML du simulateur

La classe *Simulation*, quant à elle, a la charge de mettre en route la simulation proprement dite. A partir des paramètres et des fichiers de noms que nous avons présentés, cette classe met en place l'environnement en créant le superviseur, le migrateur, tous les noeuds et tous les agents. Elle attend ensuite la terminaison de chacun de ces éléments pour mettre fin à la simulation.

A.2.2 Correspondance avec le modèle

Pour correspondre avec notre modèle, nous devons trouver sur les noeuds les différents services que nous avons décrits dans la section 5.3 (p 90). Nous allons reprendre ces différents services en montrant par quelle méthode ils sont réalisés.

Service d'exécution

Dans notre simulateur, à proprement parlé, le service d'exécution n'est pas réalisé car il n'y a pas nécessité d'arrêter et/ou de redémarrer les agents. En faisant le choix de simuler la migration d'un agent par la modification de son nœud de rattachement, les agents suspendent d'eux-mêmes leur activité durant la migration (en fonction de *dureeMigration*) et entame ensuite une nouvelle visite.

Service de voisinage

Dans le cadre de notre simulateur, nous avons intégré les possibilités offertes par le service de voisinage dans le service de migration. Ce choix est motivé par une simplification du développement.

Service de migration

Le service de migration est accessible par les deux méthodes *allerAilleurs* et *allerSur*. *allerAilleurs* permet à un agent de migrer au hasard sur un des sites appartenant au voisinage. C'est cette méthode qui prend en charge une partie des fonctionnalités du service de voisinage. *allerAilleurs* permet à un agent de se déplacer vers un site cible. Ces deux fonctions retournent le nouveau nœud de rattachement.

Les verdicts sont obtenus en vérifiant le nouveau nœud de rattachement retourné. S'il est nul, cela veut dire que la demande de migration est irréalisable. S'il ne change pas, l'agent n'a pas migrer et cela veut dire que le nœud est isolé (dans le cas de *allerSur*). Et s'il est différent, c'est que la migration est réalisée. Les cas liés aux pertes de connexion et aux échecs de relance de l'agent au cours d'une migration, sont ici ignorés car nous faisons l'hypothèse que la migration est atomique. Toute migration a réussit ou n'a pas lieu.

Service d'annuaire des agents locaux

Le service d'annuaire est réalisé par les quatre méthodes *seFaireConnaître*, *seFaireOublier*, *trouverUnAmi* et *quiEstLa*. L'annuaire utilise l'identifiant de l'agent et sa classe comme critère d'enregistrement et de recherche. Ainsi les trois premières méthodes

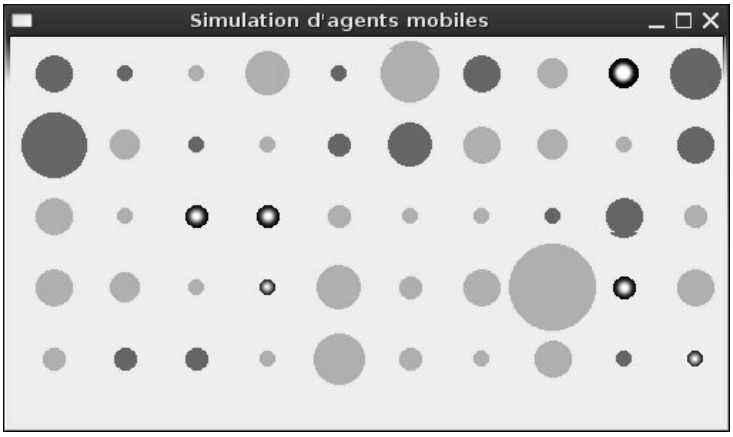


FIG. A.2 – Fenêtre de visualisation de la simulation

correspondent à la description faite dans le modèle (à savoir enregistrement, retrait et consultation simple) et la méthode `quiEstLa` permet de retrouver une liste d'agents locaux enregistrés et correspondant à un critère de recherche, dans notre cas la classe de l'agent.

Service de tableau blanc

Le service de tableau blanc n'a pas été intégralement intégré aux nœuds, mais simplement à travers les deux méthodes `enregistrer` et `quelChemin` qui permettent de mettre en place le protocole de propagation de rumeurs. Cette intégration partielle vient du fait que nous utilisons, pour le moment, uniquement le tableau blanc pour le protocole.

A.2.3 Déroulement d'une simulation

Suivi et analyse

Le déroulement d'une simulation s'effectue selon deux phases : une partie «suivi» et une partie analyse. Pour la partie suivi, nous avons réalisé une fenêtre de visualisation qui permet d'observer l'évolution du système simulé. Cette fenêtre, présentée sur la figure A.2, schématise chaque nœud par un cercle et regroupe deux types d'informations.

La première information précise de la répartition des agents. Chaque cercle à une taille variable en fonction du nombre d'agents hébergés par un site. Plus la taille est grande, plus le nombre d'agents sur le site est important. Cette première information nous permet de contrôler le déplacement des agents et de savoir si les agents se répartissent équitablement, ou non, dans l'environnement.

La deuxième information est apportée par la couleur des cercles qui permet de connaître grossièrement la taille de l'ensemble des voisins d'un site. Sur la figure on peut distinguer trois couleurs : orange, marron et gris[14]. Les couleurs orange et marron permettent de savoir si un site possède, respectivement, moins ou plus de voisins que la moyenne générale de voisinage (calculée à partir de `maxVoisins`). La couleur grise sert à identifier les sites isolés, *i.e* sans voisin. Ces trois couleurs nous permettent de contrôler que l'environnement simulé est connecté

[14]Respectivement gris clair, gris foncé et noir à centre blanc dans le cas d'un lecture du document en noir et blanc

correctement (pas de couleur dominante) et qu'un site ne reste pas isolé jusqu'au terme de la simulation.

Pour la partie analyse, lorsque la simulation est arrivée à son terme, le simulateur affiche le nombre total de rencontres effectuées, la moyenne globale de migrations par rencontre ainsi qu'un récapitulatif des paramètres définis (durée, coopération, etc.). A partir de ces deux éléments, on met en comparaison les résultats avec ceux d'autres simulations afin d'analyser les performances de notre modèle. Notons que nous avons défini un mode non-graphique (sans fenêtre de visualisation) qui nous permet de récupérer directement les résultats de la simulation.

Mode opératoire

Pour permettre une analyse de l'effet d'un paramètre à partir d'un nombre conséquent de résultats, nous avons choisi de mettre en place un mode opératoire précis. Pour cela nous avons réalisé différents scripts qui font varier un paramètre en fonction de critères précis. Pour expliquer ce mode opératoire nous prenons l'exemple de l'étude de l'adéquation de la couche d'ambiance.

Pour cette étude, nous souhaitions évaluer l'effet du pourcentage d'agents légers par rapport au nombre de sites sur l'efficacité de la couche d'ambiance. Pour cela nous avons donc fixé le nombre d'agents lourds et un intervalle pour la taille du réseau. En parcourant l'intervalle selon un pas précis, nous avons lancé à chaque fois une série de simulations où la seule différence était le nombre d'agents légers.

Pour cela, le script crée automatiquement les deux fichiers de configuration `listeDesSites` et `listeDesUsagers` par rapport au nombre de sites et d'agents lourds. Ensuite, il rajoute le nombre d'agents légers correspondant au pourcentage à évaluer, lance une simulation en mode non-graphique avec les paramètres prédéfinis et stocke les résultats obtenus dans un fichier.

Ce mode opératoire nous garantit de faire varier automatiquement un seul paramètre et d'obtenir des résultats précis qui permettent une analyse significative des effets d'un paramètre préalablement choisi.

Agents Mobiles Coopérants pour les Environnements Dynamiques

Résumé

A partir de l'étude de la mobilité dans les paradigmes de programmation pour les systèmes distribués et des différents architectures réseaux, et principalement celles sans-fils, nous avons pu identifier le problème de la gestion des unités mobiles (PDA, smartphone, ...) lorsque l'on souhaite les intégrer dans Internet. Leurs déplacements introduisent un fort dynamisme matériel qui ne permet plus d'utiliser les techniques classiques d'un internet et d'obtenir un système gérant globalement la localisation de toutes les unités.

Cette absence de gestion globale remet en cause les méthodes classiques de conception fondées sur un système offrant une représentation stable de l'environnements. Dans ce contexte, nous avons étudié la conception basée sur les agents mobiles, programmes se déplaçant de site en site de manière autonome, afin de démontrer leur utilité dans des environnements dynamiques à l'échelle d'Internet, et ce, en l'absence d'un système capable de gérer la localisation globale.

Mots-clés: Agents Mobiles, systèmes répartis, réseaux ad hoc, autonomie, adaptabilité

Abstract

From the study of programming paradigms used in distributed systems and recent network architectures, especially wireless ones, we distinguish the problem of mobile unit management (PDA, smartphone, ...) when they are involved in the Internet. Their mobility introduces a high physical dynamism which leads to reconsider design patterns used in an intranet. Such systems do not allow to provide a global view of the distribution.

This absence of global view implies to revisit classical design approaches based upon a system supplying a stable context representation. Therefore, we have studied a design approach based upon mobile agents, namely programs moving from site to site in an autonomous way. We demonstrate their usefulness in such dynamic environments at large scale in the Internet, in which there exits non global location service.

Keywords: Mobile agent, distributed systems, ad hoc networks, autonomy, adaptability